# Seahorse

## Guide de l'utilisateur



v1.2 José Facundo Maldonado

# Seahorse
# Guide de l'utilisateur
pour José Facundo Maldonado

# Index

# 1 - Introduction

Seahorse est une bibliothèque JavaScript, licencié comme logiciel libre, créé pour simplifier l'utilisation des formulaires, en particulier la validation d'entre eux. Fournit des fonctions de validation, la conversion et le comportement pour le traitement des dates, des heures, des nombres, des textes et des adresses e-mail ou des URL.

Toutes les fonctions sont hautement configurables, permettant de spécifier la plage des valeurs valides, des caractères illégaux, les formats et, dans le cas des comportements, des réponses aux événements de perte de concentration et de touche pressée.

Il peut être utilisé avec n'importe quel framework JavaScript, cependant, a un plugin pour une utilisation avec jQuery.

Seahorse est sous licence LGPL v3, ce qui signifie que vous le pouvez utiliser pour développer des projets open source et des projets commerciaux. Toutefois, dans les deux cas, vous devez publier, sous une licence compatible avec la LGPL, tous les œuvres dérivées ou des modifications que vous apportez à la bibliothèque.

## 2 - Validation

Les fonctions de validation sont chargés de vérifier si une chaîne représente ou non un type de données particulier. En plus du texte, les fonctions reçoivent certaines valeurs, comme paramètres, qui déterminent les critères selon lesquels la validation est effectuée.

### 2.1 - isNumber()

La fonction `isNumber()` vérifie si une chaîne de caractères représente un nombre. Il reçoit en tant que paramètres la chaîne, le caractère utilisé comme séparateur décimal et le caractère utilisé comme séparateur de groupes (également connu sous le séparateur des milliers).

```
Seahorse.isNumber("1,000,000.00", '.', ',')  ➡ true
Seahorse.isNumber("1 000 000.00", ',', '.')  ➡ false
Seahorse.isNumber("1 000 000.00", ',', ' ')  ➡ true
Seahorse.isNumber("1.000.000,00", ',', '.')  ➡ true
```

### 2.2 - isInteger()

La fonction `isInteger()` vérifie si une chaîne de caractères représente un nombre enteir. Il reçoit en tant que paramètres la chaîne et le caractère utilisé comme séparateur de groupes (également connu sous le séparateur des milliers).

```
Seahorse.isInteger("1,000,000", ',')  ➡ true
Seahorse.isInteger("1 000 000", ',')  ➡ false
Seahorse.isInteger("1 000 000", ' ')  ➡ false
Seahorse.isInteger("1.000.000", '.')  ➡ true
```

### 2.3 - isNumeric()

La fonction `isNumeric()` vérifie si une chaîne de caractères contient uniquement des chiffres.

```
Seahorse.isNumeric("1234")  ➡ true
Seahorse.isNumeric("1,234")  ➡ false
```

### 2.4 - isAlphabetical()

La fonction `isAlphabetical()` vérifie si une chaîne de caractères contient uniquement des lettres.

```
Seahorse.isAlphabetical("abcdñ")  ➡ true
Seahorse.isAlphabetical("abcñ123")  ➡ false
```

### 2.5 - isAlphanumeric()

La fonction `isAlphanumeric()` vérifie si une chaîne de caractères contient uniquement des lettres et des chiffres.

```
Seahorse.isAlphanumeric("abcdñ123")  ➡ true
Seahorse.isAlphanumeric("abcñ-123")  ➡ false
```

### 2.6 - isAlphabeticalAscii()

La fonction `isAlphabeticalAscii()` vérifie si une chaîne de caractères contient uniquement des lettres du latin basique.

```
Seahorse.isAlphabeticalAscii("abcd")    ➡ true
Seahorse.isAlphabeticalAscii("abcdñ")   ➡ false
```

### 2.7 - isAlphanumericAscii()

La fonction `isAlphanumericAscii()` vérifie si une chaîne de caractères contient uniquement des chiffres et des lettres du latin basique.

```
Seahorse.isAlphanumericAscii("abc123")    ➡ true
Seahorse.isAlphanumericAscii("abcñ123")   ➡ false
```

### 2.8 - isAsciiText()

La fonction `isAsciiText()` vérifie si tous les caractères de une chaîne appartiennent à la codification ASCII.

```
Seahorse.isAsciiText("abcd (123) @#")     ➡ true
Seahorse.isAsciiText("abcd (123) @#ñá")   ➡ false
```

### 2.9 - isIPv4()

La fonction `isIPv4()` vérifie si une chaîne de caractères représente une adresse IP version 4.

```
Seahorse.isIPv4("192.168.0.1")    ➡ false
Seahorse.isIPv4("192.168.0.256")  ➡ false
```

### 2.10 - isIPv6()

La fonction `isIPv6()` vérifie si une chaîne de caractères représente une adresse IP version 6.

```
Seahorse.isIPv6("11:22:33:44:55:66:77:88")  ➡ true
Seahorse.isIPv6("11:22:33:44:55:::")        ➡ true
```

### 2.11 - isEmail()

La fonction `isEmail()` vérifie si une chaîne de caractères représente une adresse d'e-mail.

```
Seahorse.isEmail("test@test.com")  ➡ true
Seahorse.isEmail("test@test")      ➡ false
```

### 2.12 - isHttp()

La fonction `isHttp()` vérifie si une chaîne de caractères représente une adresse HTTP.

```
Seahorse.isHttp("http://www.test.com")  ➡ true
Seahorse.isHttp("www.test.com")         ➡ false
```

### 2.13 - isFtp()

La fonction `isFtp()` vérifie si une chaîne de caractères représente une adresse FTP.

```
Seahorse.isFtp("ftp://ftp.test.com")  ➡ true
Seahorse.isFtp("ftp.test.com")        ➡ false
```

## 2.14 - isDate()

La fonction `isDate()` vérifie si une chaîne de caractères représente une date. Il reçoit en tant que paramètres la chaîne et le format de la date, composé avec la combination de:

`d` - jour du mois (sans zéro à gauche)
`dd` - jour du mois (deux chiffres)
`m` - mois de l'anné (sans zéro à gauche)
`mm` - mois de l'anné (deux chiffres)
`yy` - anné (deux chiffres)
`yyyy` - anné (quattre chiffres)

Le format est utilisé seulement pour determiner l'orden des chiffres. Pour ça, est équivalent, par example, utiliser comme format `dd/mm/yyyy` en tant que `dd-mm-yyyy`.

```
Seahorse.isDate("31/07/2010","dd/mm/yyyy")  ➡ true
Seahorse.isDate("31-07-2010","dd/mm/yyyy")  ➡ true
Seahorse.isDate("07/31/2010","mm/dd/yyyy")  ➡ true
Seahorse.isDate("07 31 2010","mm/dd/yyyy")  ➡ true
Seahorse.isDate("07/32/2010","mm/dd/yyyy")  ➡ false
Seahorse.isDate("06/31/2010","mm/dd/yyyy")  ➡ false
```

## 2.15 - isTime()

La fonction `isDate()` vérifie si une chaîne de caractères représente une heure. Il reçoit en tant que paramètres la chaîne et le format de l'heure, composé avec la combinaison de:

`s` - secondes (sans zéro à gauche)
`ss` - secondes (deux chiffres)
`m` - minutes (sans zéro à gauche)
`mm` - minutes (deux chiffres)
`h` - heures (sans zéro à gauche)
`hh` - heures (deux chiffres)

Le format est utilisé seulement pour determiner l'orden des chiffres. Pour ça, est équivalent, par example, utiliser comme format `hh:mm:ss` en tant que `hh-mm-ss`.

```
Seahorse.isTime("06:20:22","hh:mm:ss")  ➡ true
Seahorse.isTime("06-20-22","hh:mm:ss")  ➡ true
Seahorse.isTime("22:20:06","s:m:h")     ➡ true
Seahorse.isTime("22 20 06","s:m:h")     ➡ true
Seahorse.isTime("22:20:60","hh:mm:ss")  ➡ false
Seahorse.isTime("24:20:22","hh:mm:ss")  ➡ false
```

## 3 - Convertion

Les fonctions de conversion sont chargés d'analyser une chaîne de texte à convertir en un autre type de données ou de donner un format particulier. En plus du texte, les fonctions reçoivent, en tant que paramètres, des valeurs qui déterminent les critères selon lesquels l'analyse et la conversion sont effectuées

### 3.1 - parseNumber()

La fonction `parseNumber()` transforme une chaîne de caractères dans un chiffre. Il reçoit en tant que paramètres la chaîne, le caractère utilisé comme séparateur décimal et le caractère utilisé comme séparateur de groupes (également connu sous le séparateur des milliers).

```
Seahorse.parseNumber("1,234.5", '.', ',')      ➡ 1234.5
Seahorse.parseNumber("1,234.5", ',', '.')      ➡ 1.2345
Seahorse.parseNumber("1234 5", ',', '.')       ➡ 1234
Seahorse.parseNumber("1234sdfg5", ',', '.')    ➡ 1234
```

### 3.2 - parseInteger()

La fonction `parseInteger()` transforme une chaîne de caractères dans un entier. Il reçoit en tant que paramètres la chaîne et le caractère utilisé comme séparateur de groupes (également connu sous le séparateur des milliers).

```
Seahorse.parseInteger("1,234", ',')     ➡ 1234
Seahorse.parseInteger("1.234.5", '.')   ➡ 12345
Seahorse.parseInteger("1 234", ' ')     ➡ 1234
Seahorse.parseInteger("1 234", ',')     ➡ 1
```

### 3.3 - parseIPv4()

La fonction `parseIPv4()` transforme une chaîne de caractères représentant une adresse IP version 4 en un array de quatre entiers.

```
Seahorse.parseIPv4("192.168.0.1")      ➡ 192,168,0,1
Seahorse.parseIPv4("192.168.0.256")    ➡ null
Seahorse.parseIPv4("192 168 0 1")      ➡ null
```

### 3.4 - parseIPv6()

La fonction `parseIPv6()` transforme une chaîne de caractères représentant une adresse IP version 6 en un array de huit numéros hexadécimaux.

```
Seahorse.parseIPv6("11:22:33:44:55:66:77:88")    ➡ 0x0011,0x0022,0x0033,0x0044,
                                                   0x0055,0x0066,0x0077,0x0088
Seahorse.parseIPv6("11:22:33:44:55:66:77:FFFG")  ➡ null
Seahorse.parseIPv6("11 22 33 44 55 66 77 88")    ➡ null
```

### 3.5 - parseDate()

La fonction parseDate() reçoit une chaîne de caractères représentant une date et la transforme selon un format particulier. Il reçoit en tant que paramètres la chaîne, un booléen indiquant si les champs vides doivent être remplis avec la date actuelle et le format de la date, composé avec la combinaison de:

- `d` - jour du mois (sans zéro à gauche)
- `dd` - jour du mois (deux chiffres)
- `m` - mois de l'année (sans zéro à gauche)

`mm` - mois de l'année (deux chiffres)
`yy` - année (deux chiffres)
`yyyy` - année (quatre chiffres)

```
Seahorse.parseDate("31/07/2010","dd/mm/yyyy")          ➡ 31/07/2010
Seahorse.parseDate("31-07-2010","dd/mm/yyyy")          ➡ 31/07/2010
Seahorse.parseDate(" 07/31 (2010)","mm/dd/yyyy")       ➡ 07/31/2010
Seahorse.parseDate("07/32/2010","mm/dd/yyyy")          ➡ null
Seahorse.parseDate("07/31","mm/dd/yyyy")               ➡ null
Seahorse.parseDate("07/31","mm/dd/yyyy",true)          ➡ 07/31/2010
Seahorse.parseDate("07","mm/dd/yyyy",false)            ➡ null
Seahorse.parseDate("07","mm/dd/yyyy",true)             ➡ 07/20/2010
```

## 3.6 - parseTime()

La fonction `parseTime()` reçoit une chaîne de caractères représentant une heure et la transforme selon un format particulier. Il reçoit en tant que paramètres la chaîne, un booléen indiquant si les champs vides doivent être remplis avec l'heure actuelle et le format de l'heure, composé avec la combinaison de:

`s` - secondes (sans zéro à gauche)
`ss` - secondes (deux chiffres)
`m` - minutes (sans zéro à gauche)
`mm` - minutes (deux chiffres)
`h` - heures (sans zéro à gauche)
`hh` - heures (deux chiffres)

```
Seahorse.parseTime("02:05:33","hh:mm:ss")          ➡ 02:05:33
Seahorse.parseTime("02-05-33","hh:mm:ss")          ➡ 02:05:33
Seahorse.parseTime(" 02/05 (33)","hh:mm:ss")       ➡ 02:05:33
Seahorse.parseTime("02:60:33","hh:mm:ss")          ➡ null
Seahorse.parseTime("02:05","hh:mm:ss")             ➡ null
Seahorse.parseTime("02:05","hh:mm:ss",true)        ➡ 02:05:12
Seahorse.parseTime("07","hh:mm:ss",false)          ➡ null
Seahorse.parseTime("07","hh:mm:ss",true)           ➡ 07:50:12
```

# 4 - Comportement

Les comportements permettre la validation en temps réel des champs des formulaires par les événements de la perte de focus et une touche pressée. Les comportements sont affectées avec des fonctions différentes que reçoivent, comme paramètres, le identificateur ou le référence du champ, les options pour la validation et les options de réponse.

Les options de validation déterminent comment le champ est validé et sont similaires aux paramètres reçus par les fonctions de validation. Les options de réponse déterminent les actions à entreprendre quand il détecte que le champ a une valeur valide ou invalide.

## 4.1 - Textes

Pour définir les entrées qui acceptent des données textuelles, il existe des méthodes `text()`, qui par elle-même n'impose aucune restriction, `alphanumeric()`, qui n'accepte que les lettres et chiffres (ne permet pas de symboles ou de ponctuation), `alphabetical()`, qui accepte uniquement les lettres, et `numeric()`, qui accepte uniquement des nombres.

As validation options, all the functions accept the parameters: `notEmpty`, `minLength`, `maxLength`, `asciiCharacters`, `requiredCharacters`, `forbiddenCharacters`, `allowedCharacters` and `aditionalValidation`.

Comme options de validation, toutes les fonctions d'acceptent les paramètres: `notEmpty`, `minLength`, `maxLength`, `asciiCharacters`, `requiredCharacters`, `forbiddenCharacters`, `allowedCharacters` et `aditionalValidation`.

`notEmpty` permet d'indiquer si un champ ne peut être laissé en blanc ou non. La valeur par défaut est faux (les champs peuvent être laissés vides).

```
<html>

<head>
<title>Seahorse example</title>

<style type="text/css">
   .ok { background-color:#f0fff0; }
   .error { background-color:#fff0f0; }
</style>

<script type="text/javascript" src="js/seahorse-1.2.js"></script>
<script type="text/javascript">
   function init()
      {
      Seahorse.text("textNe1", {}, {"okClass": "ok", "errorClass": "error"});
      document.getElementById("textNe1").seahorse.verify();

      Seahorse.text("textNe2",  {"notEmpty":  false},  {"okClass":  "ok",  "errorClass":
"error"});
      document.getElementById("textNe2").seahorse.verify();

      Seahorse.text("textNe3",  {"notEmpty":  true},  {"okClass":  "ok",  "errorClass":
"error"});
      document.getElementById("textNe3").seahorse.verify();
      }
</script>
</head>

<body onload="javascript:init();">

   <form action="" method="GET">
   <center><table>
      <tr>
         <td>text()</td>
         <td></td>
         <td><input type="text" id="textNe1" value=""/></td>
      </tr>
      <tr>
         <td>text()</td>
```

```
        <td>notEmpty = true</td>
        <td><input type="text" id="textNe2" value=""/></td>
      </tr>
      <tr>
        <td>text()</td>
        <td>notEmpty = false</td>
        <td><input type="text" id="textNe3" value=""/></td>
      </tr>
   </table></center>
   </form>

</body>
</html>
```

`minLength` et `maxLength` vous permettent de définir le nombre minimal et maximal de caractères qui peut avoir une entrée.

```
<html>

<head>
<title>Seahorse example</title>

<style type="text/css">
   .ok { background-color:#f0fff0; }
   .error { background-color:#fff0f0; }
</style>

<script type="text/javascript" src="js/seahorse-1.2.js"></script>
<script type="text/javascript">
   function init()
      {
      Seahorse.text("textMl1", {"notEmpty": true, "minLength" : 3},
                {"okClass": "ok", "errorClass": "error"});
      document.getElementById("textMl1").seahorse.verify();

      Seahorse.text("textMl2",  {"maxLength"  :  6},  {"okClass":  "ok",  "errorClass":
"error"});
      document.getElementById("textMl2").seahorse.verify();
      }
</script>
</head>

<body onload="javascript:init();">

   <form action="" method="GET">
   <center><table class="cuadro_ejemplo">
      <tr>
        <td>text()</td>
        <td>notEmpty = true, minLength = 3</td>
        <td><input type="text" id="textMl1" value=""/></td>
      </tr>
      <tr>
        <td>text()</td>
        <td>maxLengt = 6</td>
        <td><input type="text" id="textMl2" value=""/></td>
      </tr>
   </table></center>
   </form>

</body>
</html>
```

`asciiCharacters` permet d'indiquer si le champ seulement permettra l'entrée des caractères ASCII, plutôt que de permettre à tous les caractères Unicode. Par défaut, tous les caractères Unicode sont autorisés.

```
<html>
```

```
<head>
<title>Seahorse example</title>

<style type="text/css">
   .ok { background-color:#f0fff0; }
   .error { background-color:#fff0f0; }
</style>

<script type="text/javascript" src="js/seahorse-1.2.js"></script>
<script type="text/javascript">
   function init()
      {
      Seahorse.text("textAc1", {}, {"okClass": "ok", "errorClass": "error"});
      document.getElementById("textAc1").seahorse.verify();

      Seahorse.text("textAc2",    {"asciiCharacters":    false},    {"okClass":    "ok",
"errorClass": "error"});
      document.getElementById("textAc2").seahorse.verify();

      Seahorse.text("textAc3", {"asciiCharacters": true}, {"okClass": "ok", "errorClass":
"error"});
      document.getElementById("textAc3").seahorse.verify();
      }
</script>
</head>

<body onload="javascript:init();">

   <form action="" method="GET">
   <center><table class="cuadro_ejemplo codigo">
      <tr>
         <td>text()</td>
         <td></td>
         <td><input type="text" id="textAc1" value=""/></td>
      </tr>
      <tr>
         <td>text()</td>
         <td>asciiCharacters = false</td>
         <td><input type="text" id="textAc2" value=""/></td>
      </tr>
      <tr>
         <td>text()</td>
         <td>asciiCharacters = true</td>
         <td><input type="text" id="textAc3" value=""/></td>
      </tr>
   </table></center>
   </form>

</body>
</html>
```

`requiredCharacters` vous permet de spécifier les caractères que doivent être inscrit à l'entrée pour être considéré valide.

```
<html>

<head>
<title>Seahorse example</title>

<style type="text/css">
   .ok { background-color:#f0fff0; }
   .error { background-color:#fff0f0; }
</style>

<script type="text/javascript" src="js/seahorse-1.2.js"></script>
<script type="text/javascript">
   function init()
```

```
      {
      Seahorse.text("textRc1", {"requiredCharacters": "_"},
                {"okClass": "ok", "errorClass": "error"});
      document.getElementById("textRc1").seahorse.verify();

      Seahorse.alphanumeric("textRc2", {"requiredCharacters": "a1"},
                {"okClass": "ok", "errorClass": "error"});
      document.getElementById("textRc2").seahorse.verify();

      Seahorse.alphabetical("textRc3",      {"requiredCharacters":      ["a".charCodeAt(0),
"b".charCodeAt(0)]},
                {"okClass": "ok", "errorClass": "error"});
      document.getElementById("textRc3").seahorse.verify();
      }
</script>
</head>

<body onload="javascript:init();">

   <form action="" method="GET">
   <center><table class="cuadro_ejemplo codigo">
      <tr>
         <td>text()</td>
         <td>requiredCharacters = "_"</td>
         <td><input type="text" id="textRc1" value=""/></td>
      </tr>
      <tr>
         <td>alphanumeric()</td>
         <td>requiredCharacters = "a1"</td>
         <td><input type="text" id="textRc2" value=""/></td>
      </tr>
      <tr>
         <td>alphabetical()</td>
         <td>requiredCharacters = ["a", "b"]</td>
         <td><input type="text" id="textRc3" value=""/></td>
      </tr>
   </table></center>
   </form>

</body>
</html>
```

`forbiddenCharacters` permet d'indiquer les caractères qui ne doivent pas être déposés pour que l'entrée soit considérée valide.

```
<html>

<head>
<title>Seahorse example</title>

<style type="text/css">
   .ok { background-color:#f0fff0; }
   .error { background-color:#fff0f0; }
</style>

<script type="text/javascript" src="js/seahorse-1.2.js"></script>
<script type="text/javascript">
   function init()
      {
      Seahorse.text("textFc1", {"forbiddenCharacters": "_"},
              {"okClass": "ok", "errorClass": "error"});
      document.getElementById("textFc1").seahorse.verify();

      Seahorse.alphanumeric("textFc2", {"forbiddenCharacters": "a1"},
              {"okClass": "ok", "errorClass": "error"});
      document.getElementById("textFc2").seahorse.verify();

      Seahorse.alphabetical("textFc3",      {"forbiddenCharacters":      ["a".charCodeAt(0),
```

```
"b".charCodeAt(0)]},
             {"okClass": "ok", "errorClass": "error"});
     document.getElementById("textFc3").seahorse.verify();
     }
</script>
</head>

<body onload="javascript:init();">

   <form action="" method="GET">
   <center><table class="cuadro_ejemplo codigo">
      <tr>
         <td>text()</td>
         <td>forbiddenCharacters = "_"</td>
         <td><input type="text" id="textFc1" value=""/></td>
      </tr>
      <tr>
         <td>alphanumeric()</td>
         <td>forbiddenCharacters = "a1"</td>
         <td><input type="text" id="textFc2" value=""/></td>
      </tr>
      <tr>
         <td>alphabetical()</td>
         <td>forbiddenCharacters = ["a", "b"]</td>
         <td><input type="text" id="textFc3" value=""/></td>
      </tr>
   </table></center>
   </form>

</body>
</html>
```

`allowedCharacters` permet d'indiquer un groupe de caractères qu'un champ peut avoir et encore être considéré valide. Ce paramètre permet, par exemple, qu'un champ de type alphabetical accepte quelques signes de ponctuation.

```
<html>

<head>
<title>Seahorse example</title>

<style type="text/css">
   .ok { background-color:#f0fff0; }
   .error { background-color:#fff0f0; }
</style>

<script type="text/javascript" src="js/seahorse-1.2.js"></script>
<script type="text/javascript">
   function init()
      {
      Seahorse.alphanumeric("textAllc1", {"allowedCharacters": " "},
             {"okClass": "ok", "errorClass": "error"});
      document.getElementById("textAllc1").seahorse.verify();

      Seahorse.alphabetical("textAllc2", {"allowedCharacters": "123"},
             {"okClass": "ok", "errorClass": "error"});
      document.getElementById("textAllc2").seahorse.verify();

      Seahorse.numeric("textAllc3",       {"allowedCharacters":       ["a".charCodeAt(0),
"b".charCodeAt(0)]},
             {"okClass": "ok", "errorClass": "error"});
      document.getElementById("textAllc3").seahorse.verify();
      }
</script>
</head>

<body onload="javascript:init();">
```

```
   <form action="" method="GET">
   <center><table class="cuadro_ejemplo codigo">
      <tr>
         <td>alphanumeric()</td>
         <td>allowedCharacters = " "</td>
         <td><input type="text" id="textAllc1" value=""/></td>
      </tr>
      <tr>
         <td>alphabetical()</td>
         <td>allowedCharacters = "123"</td>
         <td><input type="text" id="textAllc2" value=""/></td>
      </tr>
      <tr>
         <td>numeric()</td>
         <td>allowedCharacters = ["a", "b"]</td>
         <td><input type="text" id="textAllc3" value=""/></td>
      </tr>
   </table></center>
   </form>

</body>
</html>
```

`aditionalValidation` permets de spécifier une fonction pour effectuer une vérification supplémentaire pour déterminer si le champ est valide ou non.

```
<html>

<head>
<title>Seahorse example</title>

<style type="text/css">
   .ok { background-color:#f0fff0; }
   .error { background-color:#fff0f0; }
</style>

<script type="text/javascript" src="js/seahorse-1.2.js"></script>
<script type="text/javascript">
   function init()
      {
      Seahorse.text("textAv1", {"aditionalValidation": oddLength},
         {"okClass": "ok", "errorClass": "error"});
      document.getElementById("textAv1").seahorse.verify();

      Seahorse.text("textAv2", {"aditionalValidation": evenLength},
         {"okClass": "ok", "errorClass": "error"});
      document.getElementById("textAv2").seahorse.verify();
      }

   function oddLength(element)
      { return element.value.length%2 == 1;}

   function evenLength(element)
      { return element.value.length%2 == 0;}
</script>
</head>

<body onload="javascript:init();">

   <form action="" method="GET">
   <center><table class="cuadro_ejemplo codigo">
      <tr>
         <td>text()</td>
         <td>aditionalValidation = oddLength()</td>
         <td><input type="text" id="textAv1" value=""/></td>
      </tr>
      <tr>
         <td>text()</td>
```

```
      <td>aditionalValidation = evenLenght()</td>
      <td><input type="text" id="textAv2" value=""/></td>
    </tr>
  </table></center>
  </form>

</body>
</html>
```

## 4.2 - Nombres

Pour configurer des champs numériques il faut utiliser les fonctions `number()`, pour les numéros de tous types, et `integer()`, pour les entiers.

Comme les options de validation, les fonctions acceptent les paramètres: `groupingCharacter`, `decimalCharacter`, `minValue`, `maxValue`, `notEmpty` et `aditionalValidation`.

`decimalCharacter` et `groupingCharacter` permettent de définir les caractères que seront utilisés, respectivement, comme séparateur décimal et caractère de groupement (ou séparateur de milliers).

```
<html>

<head>
<title>Seahorse example</title>

<style type="text/css">
  .ok { background-color:#f0fff0; }
  .error { background-color:#fff0f0; }
</style>

<script type="text/javascript" src="js/seahorse-1.2.js"></script>
<script type="text/javascript">
  function init()
    {
    Seahorse.number( "number21", {"decimalCharacter": '.', "groupingCharacter" : ','},
            {"okClass": "ok", "errorClass": "error"} );
    document.getElementById("number21").seahorse.verify();

    Seahorse.number( "number22", {"decimalCharacter": ',', "groupingCharacter" : '.'},
            {"okClass": "ok", "errorClass": "error"} );
    document.getElementById("number22").seahorse.verify();

    Seahorse.number( "number23", {"decimalCharacter": '.', "groupingCharacter" : ' '},
            {"okClass": "ok", "errorClass": "error"} );
    document.getElementById("number23").seahorse.verify();
    }
</script>
</head>

<body onload="javascript:init();">

  <form action="" method="GET">
  <center><table class="cuadro_ejemplo">
    <tr>
      <td>number()</td>
      <td>decimalCharacter = '.', groupingCharacter = ','</td>
      <td><input type="text" id="number21" value=""/></td>
    </tr>
    <tr>
      <td>number()</td>
      <td>decimalCharacter = ',', groupingCharacter = '.'</td>
      <td><input type="text" id="number22" value=""/></td>
    </tr>
    <tr>
      <td>number()</td>
      <td>decimalCharacter = '.', groupingCharacter = ' '</td>
      <td><input type="text" id="number23" value=""/></td>
    </tr>
  </table></center>
```

```
   </form>

</body>
</html>
```

`minValue` et `maxValue` définissent les valeurs minimales et maximales que peut avoir un champ.

```
<html>

<head>
<title>Seahorse example</title>

<style type="text/css">
   .ok { background-color:#f0fff0; }
   .error { background-color:#fff0f0; }
</style>

<script type="text/javascript" src="js/seahorse-1.2.js"></script>
<script type="text/javascript">
   function init()
      {
      Seahorse.number( "numberMv1",
         {"decimalCharacter": '.', "groupingCharacter" : ',',
         "minValue" : -10.5, "maxValue" : 10.5},
         {"okClass": "ok", "errorClass": "error"} );
      document.getElementById("numberMv1").seahorse.verify();

      Seahorse.number( "numberMv2",
         {"decimalCharacter": '.', "groupingCharacter" : ',',
         "minValue" : 0, "maxValue" : 10.5},
         {"okClass": "ok", "errorClass": "error"} );
      document.getElementById("numberMv2").seahorse.verify();

      Seahorse.number( "numberMv3",
         {"decimalCharacter": '.', "groupingCharacter" : ',',
         "minValue" : -10.5, "maxValue" : 0},
         {"okClass": "ok", "errorClass": "error"} );
      document.getElementById("numberMv3").seahorse.verify();

      Seahorse.integer( "numberMv4",
         {"groupingCharacter" : ',', "minValue" : -10, "maxValue" : 10},
         {"okClass": "ok", "errorClass": "error"} );
      document.getElementById("numberMv4").seahorse.verify();

      Seahorse.integer( "numberMv5",
         {"groupingCharacter" : ',', "minValue" : 0, "maxValue" : 10},
         {"okClass": "ok", "errorClass": "error"} );
      document.getElementById("numberMv5").seahorse.verify();

      Seahorse.integer( "numberMv6",
         {"groupingCharacter" : ',', "minValue" : -10, "maxValue" : 0},
         {"okClass": "ok", "errorClass": "error"} );
      document.getElementById("numberMv6").seahorse.verify();
      }
</script>
</head>

<body onload="javascript:init();">

   <form action="" method="GET">
   <center><table class="cuadro_ejemplo">
      <tr>
         <td>number()</td>
         <td>
            decimalCharacter = '.', groupingCharacter = ','<br/>
            minValue = -10.5, maxValue = 10.5
         </td>
         <td><input type="text" id="numberMv1" value=""/></td>
```

```
      </tr>
      <tr>
         <td>number()</td>
         <td>
            decimalCharacter = '.', groupingCharacter = ','<br/>
            minValue = 0, maxValue = 10.5
         </td>
         <td><input type="text" id="numberMv2" value=""/></td>
      </tr>
      <tr>
         <td>number()</td>
         <td>
            decimalCharacter = '.', groupingCharacter = ','<br/>
            minValue = -10.5, maxValue = 0
         </td>
         <td><input type="text" id="numberMv3" value=""/></td>
      </tr>
      <tr>
         <td>integer()</td>
         <td>
            groupingCharacter = ','<br/>
            minValue = -10, maxValue = 10
         </td>
         <td><input type="text" id="numberMv4" value=""/></td>
      </tr>
      <tr>
         <td>integer()</td>
         <td>
            groupingCharacter = ','<br/>
            minValue = 0, maxValue = 10
         </td>
         <td><input type="text" id="numberMv5" value=""/></td>
      </tr>
      <tr>
         <td>integer()</td>
         <td>
            groupingCharacter = ','<br/>
            minValue = -10, maxValue = 0
         </td>
         <td><input type="text" id="numberMv6" value=""/></td>
      </tr>
   </table></center>
   </form>

</body>
</html>
```

`notEmpty` et `aditionalValidation` marchent de la même manière que dans le cas de textes.

## 4.3 - Dates et horaires

Pour définir les champs de date et l'heure il faut utiliser les fonctions `date()`, pour dates, et `time()`, pour horaires.

Comme options de validation, les fonctions acceptent les paramètres: `format`, `autofill`, `minValue`, `maxValue`, `notEmpty` et `aditionalValidation`.

`format` définit le format de la date ou l'heure. Le format des dates est constitué par la combination de 'd' ou 'dd' (jour d'un ou de deux chiffres), 'm' ou 'm' (mois d'un ou de deux chiffres) et 'yy' ou 'yyyy' (ans de deux ou de quatre chiffres). Le format pour les horaires est constitué par la combination de 's' ou 'ss' (secondes d'un ou de deux chiffres), 'm' ou 'mm' (minute d'un ou de deux chiffres) et 'h' ou 'hh' (heures d'un ou de deux chiffres).

```
<html>

<head>
<title>Seahorse example</title>
```

```
<style type="text/css">
   .ok { background-color:#f0fff0; }
   .error { background-color:#fff0f0; }
</style>

<script type="text/javascript" src="js/seahorse-1.2.js"></script>
<script type="text/javascript">
   function init()
      {
      Seahorse.date( "date21", {"format" : 'mm/dd/yyyy'}, {"okClass": "ok", "errorClass":
"error"} );
      document.getElementById("date21").seahorse.verify();

      Seahorse.date( "date22", {"format" : 'dd/mm/yyyy'}, {"okClass": "ok", "errorClass":
"error"} );
      document.getElementById("date22").seahorse.verify();

      Seahorse.date( "date23", {"format" : 'yyyy-mm-dd'}, {"okClass": "ok", "errorClass":
"error"} );
      document.getElementById("date23").seahorse.verify();

      Seahorse.time( "time21", {"format" : 'hh:mm:ss'}, {"okClass": "ok", "errorClass":
"error"} );
      document.getElementById("time21").seahorse.verify();

      Seahorse.time( "time22", {"format" : 'hh-mm-ss'}, {"okClass": "ok", "errorClass":
"error"} );
      document.getElementById("time22").seahorse.verify();
      }
</script>
</head>

<body onload="javascript:init();">

   <form action="" method="GET">
   <center><table class="cuadro_ejemplo">
      <tr>
         <td>date()</td>
         <td>format = mm/dd/yyyy</td>
         <td><input type="text" id="date21" value=""/></td>
      </tr>
      <tr>
         <td>date()</td>
         <td>format = dd/mm/yyyy</td>
         <td><input type="text" id="date22" value=""/></td>
      </tr>
      <tr>
         <td>date()</td>
         <td>format = yyyy-mm-dd</td>
         <td><input type="text" id="date23" value=""/></td>
      </tr>
      <tr>
         <td>time()</td>
         <td>format = hh:mm:ss</td>
         <td><input type="text" id="time21" value=""/></td>
      </tr>
      <tr>
         <td>time()</td>
         <td>format = hh-mm-ss</td>
         <td><input type="text" id="time22" value=""/></td>
      </tr>
   </table></center>
   </form>

</body>
</html>
```

`autofill` indique si les chiffres incomplètes de la date ou l'heure doivent être remplis avec la date ou

heure actuelle

```
<html>

<head>
<title>Seahorse example</title>

<style type="text/css">
   .ok { background-color:#f0fff0; }
   .error { background-color:#fff0f0; }
</style>

<script type="text/javascript" src="js/seahorse-1.2.js"></script>
<script type="text/javascript">
   function init()
      {
      Seahorse.date( "date31", {"format" : 'yyyy-mm-dd'},
         {"okClass": "ok", "errorClass": "error"} );
      document.getElementById("date31").seahorse.verify();

      Seahorse.date( "date32", {"format" : 'yyyy-mm-dd', 'autofill': true},
         {"okClass": "ok", "errorClass": "error"} );
      document.getElementById("date32").seahorse.verify();

      Seahorse.date( "date33", {"format" : 'yyyy-mm-dd', 'autofill': false},
         {"okClass": "ok", "errorClass": "error"} );
      document.getElementById("date33").seahorse.verify();

      Seahorse.time( "time31", {"format" : 'hh:mm:ss'},
         {"okClass": "ok", "errorClass": "error"} );
      document.getElementById("time31").seahorse.verify();

      Seahorse.time( "time32", {"format" : 'hh:mm:ss', 'autofill': true},
         {"okClass": "ok", "errorClass": "error"} );
      document.getElementById("time32").seahorse.verify();

      Seahorse.time( "time33", {"format" : 'hh:mm:ss', 'autofill': false},
{"okClass": "ok", "errorClass": "error"} );
      document.getElementById("time33").seahorse.verify();
      }
</script>
</head>

<body onload="javascript:init();">

   <form action="" method="GET">
   <center><table class="cuadro_ejemplo">
      <tr>
         <td>date()</td>
         <td>format = yyyy-mm-dd</td>
         <td><input type="text" id="date31" value=""/></td>
      </tr>
      <tr>
         <td>date()</td>
         <td>format = yyyy-mm-dd, autofill = true</td>
         <td><input type="text" id="date32" value=""/></td>
      </tr>
      <tr>
         <td>date()</td>
         <td>format = yyyy-mm-dd, autofill = false</td>
         <td><input type="text" id="date33" value=""/></td>
      </tr>
      <tr>
         <td>time()</td>
         <td>format = hh:mm:ss</td>
         <td><input type="text" id="time31" value=""/></td>
      </tr>
      <tr>
```

```
      <td>time()</td>
      <td>format = hh:mm:ss, autofill = true</td>
      <td><input type="text" id="time32" value=""/></td>
   </tr>
   <tr>
      <td>time()</td>
      <td>format = hh:mm:ss, autofill = false</td>
      <td><input type="text" id="time33" value=""/></td>
   </tr>
</table></center>
</form>

</body>
</html>
```

`minValue` et `maxValue` définit les valeurs minimales et maximales que peut avoir un champ.

```
<html>

<head>
<title>Seahorse example</title>

<style type="text/css">
   .ok { background-color:#f0fff0; }
   .error { background-color:#fff0f0; }
</style>

<script type="text/javascript" src="js/seahorse-1.2.js"></script>
<script type="text/javascript">
   function init()
      {
      Seahorse.date( "date41", {"format" : 'dd/mm/yyyy',
         "minValue" : "01/01/1996", "maxValue" : "07/07/2010"},
         {"okClass": "ok", "errorClass": "error"} );
      document.getElementById("date41").seahorse.verify();

      Seahorse.date( "date42", {"format" : 'dd/mm/yyyy',
         "minValue" : "01/01/2010", "maxValue" : "31/12/2010"},
         {"okClass": "ok", "errorClass": "error"} );
      document.getElementById("date42").seahorse.verify();

      Seahorse.time( "time41", {"format" : 'hh:mm:ss',
         "minValue" : "00:00:00", "maxValue" : "12:00:00"},
         {"okClass": "ok", "errorClass": "error"} );
      document.getElementById("time41").seahorse.verify();

      Seahorse.time( "time42", {"format" : 'hh:mm:ss',
         "minValue" : "12:34:56", "maxValue" : "23:59:59"},
         {"okClass": "ok", "errorClass": "error"} );
      document.getElementById("time42").seahorse.verify();
      }
</script>
</head>

<body onload="javascript:init();">

   <form action="" method="GET">
   <center><table class="cuadro_ejemplo">
      <tr>
         <td>date()</td>
         <td>format = dd/mm/yyyy, minValue = "01/01/1996", maxValue = "07/07/2010"</td>
         <td><input type="text" id="date41" value=""/></td>
      </tr>
      <tr>
         <td>date()</td>
         <td>format = dd/mm/yyyy, minValue = "01/01/2010", maxValue = "31/12/2010"</td>
         <td><input type="text" id="date42" value=""/></td>
```

```
      </tr>
      <tr>
         <td>time()</td>
         <td>format = hh:mm:ss, minValue = "00:00:00", maxValue = "12:00:00"</td>
         <td><input type="text" id="time41" value=""/></td>
      </tr>
      <tr>
         <td>time()</td>
         <td>format = hh:mm:ss, minValue = "12:34:56", maxValue = "23:59:59"</td>
         <td><input type="text" id="time42" value=""/></td>
      </tr>
   </table></center>
   </form>

</body>
</html>
```

`notEmpty` et `aditionalValidation` marchent de la même manière que dans le cas de textes.

## 4.4 - Adresses d'Internet

Le fonctions `ipAddress()`, `email()`, `http()` et `ftp()` permettent, respectivement, de définir des champs pour les adresses IP, e-mail et adresses HTTP ou FTP.

Comme options de validation, les fonctions acceptent seulement les paramètres `notEmpty` et `aditionalValidation`, sauf `ipAddress()` qu'il accepte également le paramètre `version` que peut prendre les valeurs 4 ou 6.

```
<html>

<head>
<title>Seahorse example</title>

<style type="text/css">
   .ok { background-color:#f0fff0; }
   .error { background-color:#fff0f0; }
</style>

<script type="text/javascript" src="js/seahorse-1.2.js"></script>
<script type="text/javascript">
   function init()
      {
      Seahorse.ipAddress(  "ipv42",  {"version"  :  4},  {"okClass":  "ok",  "errorClass":
"error"} );
      document.getElementById("ipv42").seahorse.verify();

      Seahorse.ipAddress(  "ipv62",  {"version"  :  6},  {"okClass":  "ok",  "errorClass":
"error"} );
      document.getElementById("ipv62").seahorse.verify();

      Seahorse.email( "email2", {}, {"okClass": "ok", "errorClass": "error"} );
      document.getElementById("email2").seahorse.verify();

      Seahorse.http( "http2", {}, {"okClass": "ok", "errorClass": "error"} );
      document.getElementById("http2").seahorse.verify();

      Seahorse.ftp( "ftp2", {}, {"okClass": "ok", "errorClass": "error"} );
      document.getElementById("ftp2").seahorse.verify();
      }
</script>
</head>

<body onload="javascript:init();">

   <form action="" method="GET">
   <center><table class="cuadro_ejemplo">
      <tr>
         <td>ipAddress()</td>
```

```
        <td>version = 4</td>
        <td><input type="text" id="ipv42" value=""/></td>
      </tr>
      <tr>
        <td>ipAddress()</td>
        <td>version = 6</td>
        <td><input type="text" id="ipv62" value=""/></td>
      </tr>
      <tr>
        <td>email()</td>
        <td></td>
        <td><input type="text" id="email2" value=""/></td>
      </tr>
      <tr>
        <td>http()</td>
        <td></td>
        <td><input type="text" id="http2" value=""/></td>
      </tr>
      <tr>
        <td>ftp()</td>
        <td></td>
        <td><input type="text" id="ftp2" value=""/></td>
      </tr>
   </table></center>
   </form>

</body>
</html>
```

## 4.5 - Options de réponse

Les options de réponse servent pour déterminer les actions sont effectuées quand il détecte que le champ a une valeur valide ou invalide. Contrairement aux options de validation, ses paramètres sont communs à toutes les fonctions de comportement: `okClass`, `errorClass`, `targetErrorClass`, `targetOkClass`, `targetId`, `hiddenElementId`, `callbackFunction`, `forbidEntrance`, `autoparse` et `errorMessage`.

Les paramètres `okClass` et `errorClass` déterminent les classes CSS à ajouter sur le champ, selon si sa valeur est valide ou invalide.

```
<html>

<head>
<title>Seahorse example</title>

<style type="text/css">
   .colorGreen { color:green; }
   .colorRed { color:red; }
   .colorBlue { color:blue; }
   .colorYellow { color:yellow; }
</style>

<script type="text/javascript" src="js/seahorse-1.2.js"></script>
<script type="text/javascript">
   function init()
      {
      Seahorse.alphabetical( "resOpts1a", {}, {"okClass": "colorGreen", "errorClass":
"colorRed",
         "forbidEntrance": false} );
      document.getElementById("resOpts1a").seahorse.verify();

      Seahorse.alphabetical( "resOpts1b", {}, {"okClass": "colorBlue", "errorClass":
"colorYellow",
         "forbidEntrance": false} );
      document.getElementById("resOpts1b").seahorse.verify();
      }
</script>
</head>
```

```
<body onload="javascript:init();">

   <form action="" method="GET">
   <center><table class="cuadro_ejemplo">
      <tr>
         <td>alphabetical()</td>
         <td>okClass = 'colorGreen', errorClass = 'colorRed'</td>
         <td><input type="text" id="resOpts1a" value=""/></td>
      </tr>
      <tr>
         <td>alphabetical()</td>
         <td>okClass = 'colorBlue', errorClass = 'colorYellow'</td>
         <td><input type="text" id="resOpts1b" value=""/></td>
      </tr>
   </table></center>
   </form>

</body>
</html>
```

Les paramètres `targetOkClass` et `targetErrorClass` déterminent les classes CSS que sont ajoutés, après le champ perd le focus, à un élément HTML selon le valeur du champ est valide ou invalide. Le paramètre target indique l'id de l'élément auquel les classes sont ajoutés.

```
<html>

<head>
<title>Seahorse example</title>

<style type="text/css">
   .colorGreen { color:green; }
   .colorRed { color:red; }
   .colorBlue { color:blue; }
   .colorYellow { color:yellow; }
</style>

<script type="text/javascript" src="js/seahorse-1.2.js"></script>
<script type="text/javascript">
   function init()
      {
      Seahorse.alphabetical( "resOpts2a", {},
         {"targetOkClass": "colorGreen", "targetErrorClass": "colorRed",
         "forbidEntrance": false, "targetId": "resOpts-targetA"} );

      Seahorse.alphabetical( "resOpts2b", {},
         {"targetOkClass": "colorBlue", "targetErrorClass": "colorYellow",
         "forbidEntrance": false, "targetId": "resOpts-targetB"} );
      }
</script>
</head>

<body onload="javascript:init();">

   <form action="" method="GET">
   <center><table class="cuadro_ejemplo">
      <tr>
         <td>alphabetical()</td>
         <td>
            targetOkClass = 'colorGreen', targetErrorClass = 'colorRed',<br/>
            targetId = 'resOpts-targetA'
         </td>
         <td><input type="text" id="resOpts2a" value=""/></td>
         <td id="resOpts-targetA">target</td>
      </tr>
      <tr>
         <td>alphabetical()</td>
         <td>
```

```
            targetOkClass = 'colorBlue', targetErrorClass = 'colorYellow',<br/>
            targetId = 'resOpts-targetB'
         </td>
         <td><input type="text" id="resOpts2b" value=""/></td>
         <td id="resOpts-targetB">target</td>
      </tr>
   </table></center>
   </form>

</body>
</html>
```

Le paramètre `hiddenElementId` indique l'ID d'un élément HTML qui peut être masqué ou affiché en fonction si le valeur du champ est valide ou invalide.

```
<html>

<head>
<title>Seahorse example</title>

<script type="text/javascript" src="js/seahorse-1.2.js"></script>
<script type="text/javascript">
   function init()
      {
      Seahorse.alphabetical( "resOpts3a", {},
         {"hiddenElementId": "resOpts-hiddenTargetA", "forbidEntrance": false} );
      document.getElementById("resOpts3a").seahorse.verify();

      Seahorse.alphabetical( "resOpts3b", {},
         {"hiddenElementId": "resOpts-hiddenTargetB", "forbidEntrance": false} );
      document.getElementById("resOpts3b").seahorse.verify();
      }
</script>
</head>

<body onload="javascript:init();">

   <form action="" method="GET">
   <center><table class="cuadro_ejemplo">
      <tr>
         <td>alphabetical()</td>
         <td>hiddenElementId = 'resOpts-hiddenTargetA'</td>
         <td><input type="text" id="resOpts3a" value=""/></td>
         <td><span id="resOpts-hiddenTargetA">target</span></td>
      </tr>
      <tr>
         <td>alphabetical()</td>
         <td>hiddenElementId = 'resOpts-hiddenTargetB'</td>
         <td><input type="text" id="resOpts3b" value=""/></td>
         <td><span id="resOpts-hiddenTargetB">target</span></td>
      </tr>
   </table></center>
   </form>

</body>
</html>
```

Le paramètre `callbackFunction` vous permet de définir une fonction qui est appelée lorsque le champ perd le focus et qui indique si le valeur est valide ou non.

```
<html>

<head>
<title>Seahorse example</title>

<script type="text/javascript" src="js/seahorse-1.2.js"></script>
<script type="text/javascript">
```

```
   function init()
      {
      Seahorse.alphabetical( "resOpts4a", {},
         {"callbackFunction":  function(elem,  valid)  {alert(valid);},  "forbidEntrance":
false} );

      Seahorse.numeric( "resOpts4b", {},
         {"callbackFunction":  function(elem,  valid)  {alert(valid);},  "forbidEntrance":
false} );
      }
</script>
</head>

<body onload="javascript:init();">

   <form action="" method="GET">
   <center><table class="cuadro_ejemplo">
      <tr>
         <td>alphabetical()</td>
         <td>callbackFunction = function(elem, valid) {alert(valid);}</td>
         <td><input type="text" id="resOpts4a" value=""/></td>
      </tr>
      <tr>
         <td>numeric()</td>
         <td>callbackFunction = function(elem, valid) {alert(valid);}</td>
         <td><input type="text" id="resOpts4b" value=""/></td>
      </tr>
   </table></center>
   </form>

</body>
</html>
```

Le paramètre `forbidEntrance` permet de spécifier si de prévenir, par l'événement `onKeyPress`, l'entrée de caractères qui cause que le valeur du champ ne soit pas valide. Par défaut, il empêche l'entrée de ces caractères.

```
<html>

<head>
<title>Seahorse example</title>

<script type="text/javascript" src="js/seahorse-1.2.js"></script>
<script type="text/javascript">
   function init()
      {
      Seahorse.numeric( "resOpts5a", {}, {"forbidEntrance": false} );

      Seahorse.numeric( "resOpts5b", {}, {"forbidEntrance": true} );
      }
</script>
</head>

<body onload="javascript:init();">

   <form action="" method="GET">
   <center><table class="cuadro_ejemplo">
      <tr>
         <td>numeric()</td>
         <td>forbidEntrance = 'false'</td>
         <td><input type="text" id="resOpts5a" value=""/></td>
      </tr>
      <tr>
         <td>numeric()</td>
         <td>forbidEntrance = 'true'</td>
         <td><input type="text" id="resOpts5b" value=""/></td>
      </tr>
   </table></center>
```

```
   </form>

</body>
</html>
```

Le paramètre `autoparse` vous permet de spécifier s'il faut convertir le contenu du champ, après qu'il perd le focus de celui-ci, pour éviter l'apparition de caractères inutiles.

```
<html>

<head>
<title>Seahorse example</title>

<script type="text/javascript" src="js/seahorse-1.2.js"></script>
<script type="text/javascript">
   function init()
      {
      Seahorse.integer( "resOpts6a", {"groupingCharacter" : ','}, {"autoparse": false} );

      Seahorse.integer( "resOpts6b", {"groupingCharacter" : ','}, {"autoparse": true} );
      }
</script>
</head>

<body onload="javascript:init();">

   <form action="" method="GET">
   <center><table class="cuadro_ejemplo">
      <tr>
         <td>integer()</td>
         <td>groupingCharacter : ',', autparse = 'false'</td>
         <td><input type="text" id="resOpts6a" value=""/></td>
      </tr>
      <tr>
         <td>integer()</td>
         <td>groupingCharacter : ',', autparse = 'true'</td>
         <td><input type="text" id="resOpts6b" value=""/></td>
      </tr>
   </table></center>
   </form>

</body>
</html>
```

Le paramètre `errorMessage` définit un message qui explique les conditions dans lesquelles une valeur inscrite dans le champ pour être considéré comme valide. Ce paramètre doit être défini quand le formulaire, dans lequel le champ est contenu, a été attribué un comportement par l'une des fonctions de cette bibliothèque.

## 4.6 - Formulaires

Le fonction `form()` permets ajouter un comportement aux formulaires, dans lequelle, avant l'envoi de le formulaire, vérifie si tous les champs ont des valeurs valides. Cette fonction prend comme paramètres l'id de la forme, la fonction qui est invoquée lorsque vous essayez d'envoyer le formulaire mais un ou plusieurs de ses champs ont des valeurs non valides et un message d'erreur.

```
<html>

<head>
<title>Seahorse example</title>

<style type="text/css">
   .ok { background-color:#f0fff0; }
   .error { background-color:#fff0f0; }
</style>
```

```
<script type="text/javascript" src="js/seahorse-1.2.js"></script>
<script type="text/javascript">
   function init()
      {
      Seahorse.text("form0_username", {"notEmpty": true},
          {"okClass": "ok", "errorClass": "error", "errorMessage": "Username required"});
      document.getElementById("form0_username").seahorse.verify();

      Seahorse.email("form0_email", {"notEmpty": true},
            {"okClass": "ok", "errorClass": "error", "errorMessage": "E-mail address
required"});
      document.getElementById("form0_email").seahorse.verify();

      Seahorse.form("form0", function(msjs, array) {alert(msjs);}, "Submit error" );
      }
</script>
</head>

<body onload="javascript:init();">

   <form action="" method="GET" id="form0">
   <center><table class="cuadro_formulario">
      <tr>
         <td align="right">Username</td>
         <td align="left"><input id="form0_username" type="text" name="username" value=""/
></td>
      </tr>
      <tr>
         <td align="right">Password</td>
         <td align="left"><input type="password" name="password" value="1234"/></td>
      </tr>
      <tr>
         <td align="right" valign="top">Gender</td>
         <td align="left">
            <input type="radio" name="gender" value="M" checked/><br/>
            <input type="radio" name="gender" value="F"/>
         </td>
      </tr>
      <tr>
         <td align="right">E-mail</td>
         <td align="left"><input id="form0_email" type="text" name="email" value=""/></td>
      </tr>
      <tr>
         <td align="right">Receive news</td>
         <td align="left"><input type="checkbox" name="news" value="true"/><br/></td>
      </tr>
      <tr>
         <td align="right" valign="top">Preferences</td>
         <td align="left"><select name="preferences" multiple="multiple">
            <option>Option A</option>
            <option>Option B</option>
              <option>Option C</option>
              <option>Option D</option>
         </select></td>
      </tr>
      <tr>
         <td colspan="2" align="right">
            <button id="form_0_btn">Send</button>
         </td>
      </tr>
   </table></center>
   </form>

</body>
</html>
```

## 4.7 - L'objet 'seahorse'

Quand un comportement est attribué à un élément, un objet est creé et stocké comme un attribut de l'élément. Cette objet a les options de validation et de réponse du comportement de l'élément, ainsi que la définition des fonctions `validate()`, `parse()` et `verify()`.

`validate()` et `verify()` vérifie si le champ a une valeur valide ou pas, avec le différence que `verify()` effectue également des actions définies par les choix de réponse, si parfois il peut être approprié de recourir à cette fonction après le chargement de la page.

`parse()`, si le champ est un nombre, un entier, une date, une heure ou une adresse IP, il retourne la valeur convertie, selon le type de donné.

Comme les comportements de Seahorse utilisent les événements onKeyUp, onBlur et onKeyPress pour mettre en œuvre la validation en temps réel, sur l'objet 'seahorse' sont stockées les fonctions liées à ces événements, que l'élément avait avant l'attribution du comportement. Ces fonctions sont invoquées après les fonctions relatives à la validation en temps réel.

# 5 - Autres fonctions

## 5.1 - Comparaisons

Les fonctions `compareDate()` et `compareTime()` reçoivent deux dates ou deux heures, respectivement, et vérifient si l'une est plus grand que l'autre. Reçoivent en tant que paramètres les dates ou les heures et format des mêmes, et retournent une valeur négative si la première date ou l'heure est inférieure à la seconde, zéro les deux son égales et un valeur positive si la première date ou heure est supérieure à la seconde.

## 5.2 - Sérialisation

La fonction `serialize()` encode toutes les valeurs des éléments d'un formulaire dans une chaîne en utilisant le codage URL ou JSON.

## 6 - jQuery

Le plugin de Seahorse pour jQuery permets d'utiliser les sélecteurs de jQuery pour attribuer des comportements ou de voir si les champs ont des valeurs valides, ce qui réduit le nombre de lignes de code et fait l'entretien plus facile.

Seahorse donne les fonctions `seaBehavior()`, qui attribue des comportements aux elements choisis, `seaForm()`, qui attribue des comportements aux formulaires choisis, `seaValidate()`, qui vérifie que tous les éléments ont des valeurs valides, et `seaVerify()` qui appelle la méthode `verify()` de tous les elements choisis.

```
<html>

<head>
<title>Seahorse example</title>

<style type="text/css">
   .ok { background-color:#f0fff0; }
   .error { background-color:#fff0f0; }
</style>

<script type="text/javascript" src="js/jquery-1.4.2.min.js"></script>
<script type="text/javascript" src="js/seahorse-1.2.js"></script>
<script type="text/javascript" src="js/seahorse.jquery-1.2.js"></script>
<script type="text/javascript">
   function init()
      {
      jQuery(".integer").seaBehavior( "integer",
         {"groupingCharacter" : ','},
         {"okClass": "ok", "errorClass": "error", "errorMessage": "Integer error"} );

      jQuery(".number").seaBehavior( "number",
         {"decimalCharacter": '.', "groupingCharacter" : ','},
         {"okClass": "ok", "errorClass": "error", "errorMessage": "Number error"} );

      jQuery(".date").seaBehavior( "date",
         {"format" : 'dd/mm/yyyy'},
         {"okClass": "ok", "errorClass": "error", "errorMessage": "Date error"} );

      jQuery(".time").seaBehavior( "time",
         {"format" : 'hh-mm-ss'},
         {"okClass": "ok", "errorClass": "error", "errorMessage": "Time error"} );

      jQuery(".alphabetical").seaBehavior( "alphabetical", {},
         {"okClass": "ok", "errorClass": "error", "errorMessage": "Alphabetical error"} );

      jQuery(".alphanumeric").seaBehavior( "alphanumeric", {},
         {"okClass": "ok", "errorClass": "error", "errorMessage": "Alphanumeric error"} );

      jQuery(".numeric").seaBehavior( "numeric", {},
         {"okClass": "ok", "errorClass": "error", "errorMessage": "Numeric error"} );

      jQuery(":input").seaVerify();
      jQuery("#formulario").seaForm(function(msjs,    array)    {alert(msjs);},    "Submit
error");
      }
</script>
</head>

<body onload="javascript:init();">

   <form action="" method="GET" id="formulario">
   <center><table class="cuadro_ejemplo">
      <tr>
         <td>number()</td>
         <td><input type="text" class="number" value="1,000.00"/></td>
      </tr>
      <tr>
```

```
        <td>integer()</td>
        <td><input type="text" class="integer" value="1,000"/></td>
      </tr>
      <tr>
        <td>date()</td>
        <td><input type="text" class="date" value="31/12/2000"/></td>
      </tr>
      <tr>
        <td>time()</td>
        <td><input type="text" class="time" value="23:59:59"/></td>
      </tr>
      <tr>
        <td>alphanumeric()</td>
        <td><input type="text" class="alphanumeric" value="Alphanumeric123"/></td>
      </tr>
      <tr>
        <td>alphabetical()</td>
        <td><input type="text" class="alphabetical" value="Alphabetical"/></td>
      </tr>
      <tr>
        <td>numeric()</td>
        <td><input type="text" class="numeric" value="1234"/></td>
      </tr>
      <tr>
        <td colspan="2" align="right">
           <input type="submit" value="Submit"/>
        </td>
      </tr>
   </table></center>
   </form>

</body>
</html>
```

# 7 - Annexe

## 7.1 - Spécification API

### 7.1.1 - Sumary

**About Seahorse**

| | |
|---|---|
| *Introduction* | Seahorse is an open source JavaScript library created for simplify the use of forms, particularly the form validation. |
| *License* | This library is licensed under the LGPL Version 3 license. |

**Validation behaviors**

| | |
|---|---|
| *Validation options* | The validation options are parameters that restrict the values, or the format, that the data of an input can have to be considered valid. |
| *Response options* | The response options are parameters that determines the courses of actions when an user enter a valid or invalid input. |

Functions

*Validation behaviors*

| | |
|---|---|
| *text()* | Gives to an element a text input behavior. |
| *alphabetical()* | Gives to an element an alphabetical input behavior. |
| *alphanumeric()* | Gives to an element an alphanumeric input behavior. |
| *numeric()* | Gives to an element a numeric input behavior. |
| *number()* | Gives to an element a float input behavior. |
| *integer()* | Gives to an element an integer input behavior. |
| *date()* | Gives to an element a date input behavior. |
| *time()* | Gives to an element a time input behavior. |
| *ipAddress()* | Gives to an element an IP address input behavior. |
| *email()* | Gives to an element an e-mail address input behavior. |
| *http()* | Gives to an element a HTTP address input behavior. |
| *ftp()* | Gives to an element a FTP address input behavior. |
| *form()* | Gives to an element a form behavior. |
| *removeBehavior()* | Removes, from a element, any behavior asigned by a function of the Seahorse's library. |

***Validation functions***

| | |
|---|---|
| *isNumber()* | Checks if a string represents a number. |
| *isInteger()* | Checks if a string represents an integer. |
| *isNumeric()* | Checks if a string contains only numbers. |
| *isAlphabetical()* | Checks if a string contains only alphabetical characters (including accents and another not Ascii characters). |
| *isAlphanumeric()* | Checks if a string contains only alphanumeric characters. |
| *isAlphabeticalAscii()* | Checks if a string contains only alphabetical ASCII characters. |
| *isAlphanumericAscii()* | Checks if a string contains only alphanumeric ASCII characters. |
| *isAsciiText()* | Checks if a string contains only ASCII characters. |
| *isIPv4()* | Checks if a string represents an IP version 4 address. |
| *isIPv6()* | Checks if a string represents an IP version 6 address. |
| *isEmail()* | Checks if a string represents an e-mail address. |
| *isHttp()* | Checks if a string represents a HTTP address. |
| *isFtp()* | Checks if a string represents a FTP address. |
| *isMonth()* | Checks if a string represents a month. |
| *isDay()* | Checks if a string represents a day of a particular month. |
| *isDate()* | Checks if string represents a date. |
| *isTime()* | Checks if string represents an instant of time. |
| *isDateFormat()* | Checks if string is a valid date format. |
| *isTimeFormat()* | Checks if string is a valid time format. |
| *validateForm()* | Checks if all the elements of a form have valid values. |
| *passFilter()* | Acording to the value of 'contains', checks if all the characters of 'cad' are contained in 'filter' or if neither of the characters are contained. |

***Parsing functions***

| | |
|---|---|
| *parseNumber()* | Parses a string and returns an number. |
| *parseInteger()* | Parses a string and returns a integer. |
| *parseIPv4()* | Parses a string that represents an IPv4 address and returns a array with four numbers. |
| *parseIPv6()* | Parses a string that represents an IPv6 address and returns a array with eight hexadecimal numbers. |
| *parseDate()* | Receives a string representing a date and transforms it according to the format passed as parameter. |
| *parseTime()* | Receives a string representing an instant of time and transforms it according to the format passed as parameter. |
| *filterString()* | Returns all the characters of 'cad' that are contained, or that aren't contained, in 'filter'. |

***Others***

| | |
|---|---|
| *compareDate()* | Compare two strings that represents dates. |
| *compareTime()* | Compare two strings that represents times. |
| *serialize()* | Encode the elements of a form as a string. |

### 7.1.2 - About Seahorse

*Introduction*

Seahorse is an open source JavaScript library created for simplify the use of forms, particularly the form validation.

It provides functions to validate, parse and serialize data and to add real-time validation to inputs.

Seahorse can be used alone or it can be used with jQuery, using the plugin designed for that purpose.

*License*

This library is licensed under the LGPL Version 3 license.

That means you can use Seahorse to develop commercial or open source projects, but, in both cases, you must publish, under a licence compatible with LGPL v3, any changes you make to the library.

### 7.1.3 - Validation behaviors

One of the main features of this library are the functions to assign validation behaviors to the inputs.

An input with a validation behavior can avoid the input of invalid characters or perform diferent actions (like add a CSS class to a element or invoke a JavaScript function) when a user inserts valid or invalid data.

The validation behaviors functions receives as parameter the id of the element to be validated, a JSON element with the validation options and a JSON element with the response options.

*Validation options*

The validation options are parameters that restrict the values, or the format, that the data of an input can have to be considered valid.

The variables that can be defined in the JSON element of validation options are:

| | |
|---|---|
| *notEmpty* | A boolean indicating if the field can be left in blank. |
| *minLength* | The minimum length of a text input (by default, 0). |
| *maxLength* | The maximun length of a text input (by default, infinity). |
| *minValue* | The minimum value of a input (by default, minus infinity for numbers and null for dates and times). |
| *maxValue* | The maximun value of a input (by default, infinity for numbers and null for dates and times). |
| *format* | The format of a input (by default 'yyyy-mm-dd' for dates and 'hh:mm:ss' for times). |
| *autofill* | A boolean indicating if the incomplete fields of dates or times must be completed with the actual date or time. |
| *version* | The version of an IP address input (by default, 4). |
| *requiredCharacters* | A string or a array of unicode values representing the group of characters that a text input must have to be considered valid. |
| *forbiddenCharacters* | A string or a array of unicode values representing the group of characters that a text input must not have to be considered valid. |
| *allowedCharacters* | A string or a array of unicode values representing the group of characters that a text input can have and be considered valid, no matter the restrictions of his type. |
| *asciiCharacters* | A boolean indicating if a text input must have only ASCII characters (by default 'false'). |
| *decimalCharacter* | The character used as decimal character in a numeric input (by default '.'). |
| *groupingCharacter* | The character used as grouping character in a numeric input (by default ','). |

| | |
|---|---|
| *aditionalValidation* | A user's defined function that test if a field has a valid value. This function is called only if the field has been validated by the standard Seahorse's validation function. |

This variables are optionals and they are not used by all the functions, each function uses only a few variables, for example, for restrict the lenght, text() uses minLength and maxLength while number() uses minValue and maxValue.

*Response options*

The response options are parameters that determines the courses of actions when an user enter a valid or invalid input.

The variables that can be defined in the JSON element of response options are:

| | |
|---|---|
| *errorClass* | The class added to the input if the data entered is invalid. |
| *okClass* | The class added to the input if the data entered is valid. |
| *targetErrorClass* | The class added to a given element if the data entered is invalid. |
| *targetOkClass* | The class added to a given element if the data entered is valid. |
| *targetId* | The id of the element to which add the classes 'targetErrorClass' or 'targetOkClass'. |
| *hiddenElementId* | The id of the element to hide or show, depending if the data entered is valid or invalid. |
| *callbackFunction* | A function to invoke after that the input was validated. The function must receive two parameters: 'element' (the object that represents the input) and 'valid' (a boolean indicating if the data entered is valid or not). |
| *forbidEntrance* | A boolean indicating if the entrance of invalid characters must be forbidden (by default, 'true'). |
| *autoparse* | A boolean indicating if the data in the fields must be parsed in order to eliminate unnecessary characters. |
| *errorMessage* | A message explaining why the value of the input is invalid. |

This variables are optionals and can be specified for all the validation functions.

### 7.1.4 - Functions

*Sumary*

***Validation behaviors***

| | |
|---|---|
| text() | Gives to an element a text input behavior. |
| alphabetical() | Gives to an element an alphabetical input behavior. |
| alphanumeric() | Gives to an element an alphanumeric input behavior. |
| numeric() | Gives to an element a numeric input behavior. |
| number() | Gives to an element a float input behavior. |
| integer() | Gives to an element an integer input behavior. |
| date() | Gives to an element a date input behavior. |
| time() | Gives to an element a time input behavior. |
| ipAddress() | Gives to an element an IP address input behavior. |
| email() | Gives to an element an e-mail address input behavior. |
| http() | Gives to an element a HTTP address input behavior. |
| ftp() | Gives to an element a FTP address input behavior. |
| form() | Gives to an element a form behavior. |
| removeBehavior() | Removes, from a element, any behavior asigned by a function of the Seahorse's library. |

***Validation functions***

| | |
|---|---|
| isNumber() | Checks if a string represents a number. |
| isInteger() | Checks if a string represents an integer. |
| isNumeric() | Checks if a string contains only numbers. |
| isAlphabetical() | Checks if a string contains only alphabetical characters (including accents and another not Ascii characters). |
| isAlphanumeric() | Checks if a string contains only alphanumeric characters. |
| isAlphabeticalAscii() | Checks if a string contains only alphabetical ASCII characters. |
| isAlphanumericAscii() | Checks if a string contains only alphanumeric ASCII characters. |
| isAsciiText() | Checks if a string contains only ASCII characters. |
| isIPv4() | Checks if a string represents an IP version 4 address. |
| isIPv6() | Checks if a string represents an IP version 6 address. |
| isEmail() | Checks if a string represents an e-mail address. |
| isHttp() | Checks if a string represents a HTTP address. |

| | |
|---|---|
| isFtp() | Checks if a string represents a FTP address. |
| isMonth() | Checks if a string represents a month. |
| isDay() | Checks if a string represents a day of a particular month. |
| isDate() | Checks if string represents a date. |
| isTime() | Checks if string represents an instant of time. |
| isDateFormat() | Checks if string is a valid date format. |
| isTimeFormat() | Checks if string is a valid time format. |
| validateForm() | Checks if all the elements of a form have valid values. |
| passFilter() | Acording to the value of 'contains', checks if all the characters of 'cad' are contained in 'filter' or if neither of the characters are contained. |

***Parsing functions***

| | |
|---|---|
| parseNumber() | Parses a string and returns an number. |
| parseInteger() | Parses a string and returns a integer. |
| parseIPv4() | Parses a string that represents an IPv4 address and returns a array with four numbers. |
| parseIPv6() | Parses a string that represents an IPv6 address and returns a array with eight hexadecimal numbers. |
| parseDate() | Receives a string representing a date and transforms it according to the format passed as parameter. |
| parseTime() | Receives a string representing an instant of time and transforms it according to the format passed as parameter. |
| filterString() | Returns all the characters of 'cad' that are contained, or that aren't contained, in 'filter'. |

***Others***

| | |
|---|---|
| compareDate() | Compare two strings that represents dates. |
| compareTime() | Compare two strings that represents times. |
| serialize() | Encode the elements of a form as a string. |

*Validation behaviors*

## text()

```
text : function(  element,
                  validationOptions,
                  responseOptions    )
```

Gives to an element a text input behavior.

*Parameters*

| | |
|---|---|
| element | The element id or the element object. |
| validationOptions | A JSON element with the validation options (notEmpty, aditionalValidation, minLength, maxLength, asciiCharacters, allowedCharacters, requiredCharacters and forbiddenCharacters). |
| responseOptions | A JSON element with the response options. |

## alphabetical()

```
alphabetical : function(  element,
                          validationOptions,
                          responseOptions    )
```

Gives to an element an alphabetical input behavior.

*Parameters*

| | |
|---|---|
| element | The element id or the element object. |
| validationOptions | A JSON element with the validation options (notEmpty, aditionalValidation, minLength, maxLength, asciiCharacters, allowedCharacters, requiredCharacters and forbiddenCharacters). |
| responseOptions | A JSON element with the response options. |

### alphanumeric()

```
alphanumeric : function(  element,
                          validationOptions,
                          responseOptions    )
```

Gives to an element an alphanumeric input behavior.

*Parameters*

| | |
|---|---|
| element | The element id or the element object. |
| validationOptions | A JSON element with the validation options (notEmpty, aditionalValidation, minLength, maxLength, asciiCharacters, allowedCharacters, requiredCharacters and forbiddenCharacters). |
| responseOptions | A JSON element with the response options. |

### numeric()

```
numeric : function(  element,
                     validationOptions,
                     responseOptions    )
```

Gives to an element a numeric input behavior.

*Parameters*

| | |
|---|---|
| element | The element id or the element object. |
| validationOptions | A JSON element with the validation options (notEmpty, aditionalValidation, minLength, maxLength, asciiCharacters, allowedCharacters, requiredCharacters and forbiddenCharacters). |
| responseOptions | A JSON element with the response options. |

### number()

```
number : function(  element,
                    validationOptions,
                    responseOptions    )
```

Gives to an element a float input behavior.

*Parameters*

| | |
|---|---|
| element | The element id or the element object. |
| validationOptions | A JSON element with the validation options (notEmpty, aditionalValidation, minValue, maxValue, groupingCharacter and decimalCharacter). |
| responseOptions | A JSON element with the response options. |

### integer()

```
integer : function(  element,
                     validationOptions,
                     responseOptions    )
```

Gives to an element an integer input behavior.

*Parameters*

| | |
|---|---|
| element | The element id or the element object. |
| validationOptions | A JSON element with the validation options (notEmpty, aditionalValidation, minValue, maxValue and groupingCharacter). |

| | |
|---|---|
| responseOptions | A JSON element with the response options. |

### date()

```
date : function(  element,
                  validationOptions,
                  responseOptions     )
```

Gives to an element a date input behavior.

*Parameters*

| | |
|---|---|
| element | The element id or the element object. |
| validationOptions | A JSON element with the validation options (notEmpty, aditionalValidation, minValue, maxValue, autofill and format). |
| responseOptions | A JSON element with the response options. |

### time()

```
time : function(  element,
                  validationOptions,
                  responseOptions     )
```

Gives to an element a time input behavior.

*Parameters*

| | |
|---|---|
| element | The element id or the element object. |
| validationOptions | A JSON element with the validation options (notEmpty, aditionalValidation, minValue, maxValue, autofill and format). |
| responseOptions | A JSON element with the response options. |

### ipAddress()

```
ipAddress : function(  element,
                       validationOptions,
                       responseOptions     )
```

Gives to an element an IP address input behavior.

*Parameters*

| | |
|---|---|
| element | The element id or the element object. |
| validationOptions | A JSON element with the validation options (notEmpty, aditionalValidation and version). |
| responseOptions | A JSON element with the response options. |

### email()

```
email : function(  element,
                   validationOptions,
                   responseOptions     )
```

Gives to an element an e-mail address input behavior.

*Parameters*

| | |
|---|---|
| element | The element id or the element object. |
| validationOptions | A JSON element with the validation options (notEmpty and aditionalValidation). |
| responseOptions | A JSON element with the response options. |

### http()

```
http : function(  element,
                  validationOptions,
                  responseOptions    )
```

Gives to an element a HTTP address input behavior.

*Parameters*

| | |
|---|---|
| element | The element id or the element object. |
| validationOptions | A JSON element with the validation options (notEmpty and aditionalValidation). |
| responseOptions | A JSON element with the response options. |

### ftp()

```
ftp : function(  element,
                 validationOptions,
                 responseOptions    )
```

Gives to an element a FTP address input behavior.

*Parameters*

| | |
|---|---|
| element | The element id or the element object. |
| validationOptions | A JSON element with the validation options (notEmpty and aditionalValidation). |
| responseOptions | A JSON element with the response options. |

### form()

```
form : function(  form,
                  responseFunction,
                  errorMessage       )
```

Gives to an element a form behavior.
That means that it modifies the submit method of the form in order to validate all the inputs before submit. The 'responseFunction' parameter is the function called when the form is submited but one or more inputs, of the form, has invalid values. This function must receive two parameters: a string with the error messages of the form and the inputs and an array with all the input elements that have invalid values.

*Parameters*

| | |
|---|---|
| form | The form id or the form object. |
| responseFunction | The function called if one of more inputs of the form are invalid. |
| errorMessage | The message to be displayed if one of more inputs of the form are invalid. |

### removeBehavior()

```
removeBehavior : function(    element )
```

Removes, from a element, any behavior asigned by a function of the Seahorse's library.

*Parameters*

element The element id or the element object.

### isNumber()

```
isNumber : function(  cad,
                      ds,
                      gs   )
```

Checks if a string represents a number.

*Parameters*
cad A string.
ds The character used as digital separator.
gs The character used as grouping separator.

*Returns*
`true` if the string represents a number and `false` in the opposite case.

### isInteger()

```
isInteger : function(  cad,
                       gs   )
```

Checks if a string represents an integer.

*Parameters*
cad A string.
gs The character used as grouping separator.

*Returns*
`true` if the string represents an integer and `false` in the opposite case.

### isNumeric()

```
isNumeric : function(  cad )
```

Checks if a string contains only numbers.

*Parameters*
cad A string.

*Returns*
`true` if the string contains only numbers and `false` in the opposite case.

### isAlphabetical()

```
isAlphabetical : function(  cad )
```

Checks if a string contains only alphabetical characters (including accents and another not Ascii characters).

*Parameters*
cad A string.

*Returns*
`true` if the string contains only alphabetical characters and `false` in the opposite case.

### isAlphanumeric()

isAlphanumeric : function( cad )

Checks if a string contains only alphanumeric characters.

*Parameters*
cad A string.

*Returns*
`true` if the string contains only alphanumeric characters and `false` in the opposite case.

### isAlphabeticalAscii()

isAlphabeticalAscii : function( cad )

Checks if a string contains only alphabetical ASCII characters.

*Parameters*
cad A string.

*Returns*
`true` if the string contains only alphabetical ASCII characters and `false` in the opposite case.

### isAlphanumericAscii()

isAlphanumericAscii : function( cad )

Checks if a string contains only alphanumeric ASCII characters.

*Parameters*
cad A string.

*Returns*
`true` if the string contains only alphanumeric ASCII characters and `false` in the opposite case.

### isAsciiText()

isAsciiText : function( cad )

Checks if a string contains only ASCII characters.

*Parameters*
cad A string.

*Returns*
`true` if the string contains only ASCII characters and `false` in the opposite case.

### isIPv4()

isIPv4 : function( ip )

Checks if a string represents an IP version 4 address.

*Parameters*
ip A string.

*Returns*
`true` if the string represents an IP version 4 address and `false` in the opposite case.

### isIPv6()
isIPv6 : function( ip )

Checks if a string represents an IP version 6 address.

*Parameters*
ip A string.

*Returns*
`true` if the string represents an IP version 6 address and `false` in the opposite case.

### isEmail()
isEmail : function( mail )

Checks if a string represents an e-mail address.

*Parameters*
mail A string.

*Returns*
`true` if the string represents an e-mail address and `false` in the opposite case.

### isHttp()
isHttp : function( http )

Checks if a string represents a HTTP address.

*Parameters*
http A string.

*Returns*
`true` if the string represents a HTTP address and `false` in the opposite case.

### isFtp()
isFtp : function( ftp )

Checks if a string represents a FTP address.

*Parameters*
ftp A string.

*Returns*
`true` if the string represents a FTP address and `false` in the opposite case.

### isMonth()
isMonth : function( mes )

Checks if a string represents a month. This function considers that a month is a number that must be between 1 and 12 (while the class Date considers a month like a number between 0 and 11).

*Parameters*
mes A string.

*Returns*
`true` if the string represents a month and `false` in the opposite case.

### isDay()

```
isDay : function(  dia,
                   mes,
                   anio )
```

Checks if a string represents a day of a particular month. This function considers that a month is a number that must be between 1 and 12 (while the class Date considers a month like a number between 0 and 11).

*Parameters*
dia A string that represents a day.
mes A string that represents a month.
anio A string that represents a year.

*Returns*
`true` if 'dia' represents a day of the month 'mes' and `false` in the opposite case or if 'mes' it's not a month.

### isDate()

```
isDate : function(  cad,
                    format,
                    fill       )
```

Checks if string represents a date.

*Parameters*
cad A string that represents a date.
format A string that represents a date format.
fill A boolean that indicates if the empty fields have to be completed with the actual date.

*Returns*
`true` if the string represents a date and `false` in the opposite case.

### isTime()

```
isTime : function(  cad,
                    format,
                    fill       )
```

Checks if string represents an instant of time.

*Parameters*
cad A string that represents an instant of time.
format A string that represents a time format.
fill A boolean that indicates if the empty fields have to be completed with the actual time.

*Returns*

`true` if the string represents an instant of time and `false` in the opposite case.

### isDateFormat()

isDateFormat : function( format )

Checks if string is a valid date format.

*Parameters*
cad A string that represents a date format.

*Returns*
`true` if the string is a valid date format and `false` in the opposite case.

### isTimeFormat()

isTimeFormat : function( format )

Checks if string is a valid time format.

*Parameters*
cad A string that represents a time format.

*Returns*
`true` if the string is a valid time format and `false` in the opposite case.

### validateForm()

validateForm : function( idForm )

Checks if all the elements of a form have valid values.

*Parameters*
idForm The id of the form to be validated.

*Returns*
`true` if all the elements of the form have valid values. `false` if one or more elements of the form have invalid values.

### passFilter()

passFilter : function( cad,
filter,
contains )

Acording to the value of 'contains', checks if all the characters of 'cad' are contained in 'filter' or if neither of the characters are contained.

*Parameters*
cad A string to filter.
filter A string used as filter.
contains A boolean indicating the mode of operation.

*Returns*
`true` if all the characters of 'cad' are contained in 'filter' and `false` in the oposite case. `true` if neither of the characters of 'cad' are contained in 'filter' and `false` in the opposite case.

*Parsing functions*

### parseNumber()

```
parseNumber : function(  cad,
                         ds,
                         gs   )
```

Parses a string and returns an number.

*Parameters*
cad A string.
ds The character used as digital separator.
gs The character used as grouping separator.

Returns
A number if the string is a number or `NaN` in the opposite case.

### parseInteger()

```
parseInteger : function(  cad,
                          gs   )
```

Parses a string and returns a integer.

*Parameters*
cad A string.
gs The character used as grouping separator.

*Returns*
An integer if the string is a number or `NaN` in the opposite case.

### parseIPv4()

```
parseIPv4 : function(  ip )
```

Parses a string that represents an IPv4 address and returns a array with four numbers.

*Parameters*
*ip* A string.

*Returns*
An array of four numbers if the string represents an IPv4 address or `null` in the opposite case.

### parseIPv6()

```
parseIPv6 : function(  ip )
```

Parses a string that represents an IPv6 address and returns a array with eight hexadecimal numbers.

*Parameters*
*ip* A string.

*Returns*
An array of eight hexadecimal numbers if the string represents an IPv6 address or `null` in the opposite case.

### parseDate()

```
parseDate : function(  cad,
                       format,
                       fill       )
```

Receives a string representing a date and transforms it according to the format passed as parameter.

*Parameters*
cad A string that represents a date.
format A string that represents a valid date format.
fill A boolean that indicates if the empty fields have to be completed with the actual date.

*Returns*
A formated string that represents a date if 'cad' represents a date and 'format' is a valid date format or `null` in the opposite case.

### parseTime()

```
parseTime : function(  cad,
                       format,
                       fill       )
```

Receives a string representing an instant of time and transforms it according to the format passed as parameter.

*Parameters*
cad A string that represents an instant of time.
format A string that represents a valid time format.
fill A boolean that indicates if the empty fields have to be completed with the actual time.

*Returns*
A formated string that represents an instant of time if 'cad' represents an instant of time and 'format' is a valid time format or `null` in the opposite case.

### filterString()

```
filterString : function(  cad,
                          filter,
                          contains )
```

Returns all the characters of 'cad' that are contained, or that aren't contained, in 'filter'.

*Parameters*
cad A string to filter.
filter A string used as filter.
contains A boolean indicating the mode of operation.

*Returns*
All the characters of 'cad' that are contained in 'filter', if 'contains' is equal to `true`. All the characters of 'cad' that aren't contained in 'filter', if 'contains' is equal to `false`.

*Others*

### compareDate()

```
compareDate : function(  date1,
```

```
                                date2,
                                dateFormat )
```

Compare two strings that represents dates.

*Parameters*

| | |
|---|---|
| date1 | The first date. |
| date2 | The second date. |
| dateFormat | The date format of the two dates. |

*Returns*

A negative number if date1 < date2, a positive number if date1 > date2, zero if date1 = date2 or null in case of error.

### compareTime()

```
compareTime : function(   time1,
                          time2,
                          timeFormat  )
```

Compare two strings that represents times.

*Parameters*

| | |
|---|---|
| time1 | The first time. |
| time2 | The second time. |
| timeFormat | The date format of the two times. |

*Returns*

A negative number if time1 < time2, a positive number if time1 > time2, zero if time1 = time2 or null in case of error.

### serialize()

```
serialize : function(   form,
                        notation )
```

Encode the elements of a form as a string.

*Parameters*

| | |
|---|---|
| form | The form id or the form object. |
| codification | The notation to be used (JSON or URL) |

*Returns*

The form serialized.

## 7.2 - Exemples

### 7.2.1 - Validation

```
<html>

<head>
<title>Seahorse's examples - Validation</title>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<script type="text/javascript" src="js/seahorse-1.2.js"></script>

</head>
<body>

   <table>
      <tr>
         <td>Seahorse.isNumber("1.000.000,00", ',', '.')</td>
         <td><img src="img/icon-arrow-right.png" title="=" alt="="/></td>
         <td>
            <script type="text/javascript">
               document.write(Seahorse.isNumber("1.000.000,00", ',', '.'));
            </script>
         </td>
      </tr>
      <tr>
         <td>Seahorse.isInteger("1,000,000", ',')</td>
         <td><img src="img/icon-arrow-right.png" title="=" alt="=" /></td>
         <td>
            <script type="text/javascript">
               document.write(Seahorse.isInteger("1,000,000", ','));
            </script>
         </td>
      </tr>
      <tr>
         <td>Seahorse.isNumeric("1234")</td>
         <td><img src="img/icon-arrow-right.png" title="=" alt="="/></td>
         <td>
            <script type="text/javascript">
               document.write(Seahorse.isNumeric("1234"));
            </script>
         </td>
      </tr>
      <tr>
         <td>Seahorse.isAlphabetical("abcd&ntilde;")</td>
         <td><img src="img/icon-arrow-right.png" title="=" alt="="/></td>
         <td>
            <script type="text/javascript">
               document.write(Seahorse.isAlphabetical("abcd\u00c9"));
            </script>
         </td>
      </tr>
      <tr>
         <td>Seahorse.isAlphanumeric("abcd&ntilde;123")</td>
         <td><img src="img/icon-arrow-right.png" title="=" alt="="/></td>
         <td>
            <script type="text/javascript">
               document.write(Seahorse.isAlphanumeric("abcd\u00c9123"));
            </script>
         </td>
      </tr>
      <tr>
         <td>Seahorse.isAlphabeticalAscii("abcd")</td>
         <td><img src="img/icon-arrow-right.png" title="=" alt="="/></td>
         <td>
            <script type="text/javascript">
               document.write(Seahorse.isAlphabeticalAscii("abcd"));
            </script>
         </td>
```

```
</tr>
<tr>
   <td>Seahorse.isAlphanumericAscii("abc123")</td>
   <td><img src="img/icon-arrow-right.png" title="=" alt="="/></td>
   <td>
      <script type="text/javascript">
         document.write(Seahorse.isAlphanumericAscii("abcd"));
      </script>
   </td>
</tr>
<tr>
   <td>Seahorse.isAsciiText("abcd (123) @#")</td>
   <td><img src="img/icon-arrow-right.png" title="=" alt="="/></td>
   <td>
      <script type="text/javascript">
         document.write(Seahorse.isAsciiText("abcd"));
      </script>
   </td>
</tr>
<tr>
   <td>Seahorse.isIPv4("192.168.0.1")</td>
   <td><img src="img/icon-arrow-right.png" title="=" alt="="/></td>
   <td>
      <script type="text/javascript">
         document.write(Seahorse.isIPv4("192.168.0.1"));
      </script>
   </td>
</tr>
<tr>
   <td>Seahorse.isIPv6("11:22:33:44:55:66:77:88")</td>
   <td><img src="img/icon-arrow-right.png" title="=" alt="="/></td>
   <td>
      <script type="text/javascript">
         document.write(Seahorse.isIPv6("11:22:33:44:55:66:77:88"));
      </script>
   </td>
</tr>
<tr>
   <td>Seahorse.isEmail("test@test.com")</td>
   <td><img src="img/icon-arrow-right.png" title="=" alt="="/></td>
   <td>
      <script type="text/javascript">
         document.write(Seahorse.isEmail("test@test.com"));
      </script>
   </td>
</tr>
<tr>
   <td>Seahorse.isHttp("http://www.test.com")</td>
   <td><img src="img/icon-arrow-right.png" title="=" alt="="/></td>
   <td>
      <script type="text/javascript">
         document.write(Seahorse.isHttp("http://www.test.com"));
      </script>
   </td>
</tr>
<tr>
   <td>Seahorse.isFtp("ftp://ftp.test.com")</td>
   <td><img src="img/icon-arrow-right.png" title="=" alt="="/></td>
   <td>
      <script type="text/javascript">
         document.write(Seahorse.isFtp("ftp://ftp.test.com"));
      </script>
   </td>
</tr>
<tr>
   <td>Seahorse.isDate("31/07/2010","dd/mm/yyyy")</td>
   <td><img src="img/icon-arrow-right.png" title="=" alt="="/></td>
   <td>
```

```
            <script type="text/javascript">
               document.write(Seahorse.isDate("31/07/2010","dd/mm/yyyy"));
            </script>
         </td>
      </tr>
      <tr>
         <td>Seahorse.isTime("06:20:22","hh:mm:ss")</td>
         <td><img src="img/icon-arrow-right.png" title="=" alt="="/></td>
         <td>
            <script type="text/javascript">
               document.write(Seahorse.isTime("06:20:22","hh:mm:ss"));
            </script>
         </td>
      </tr>
   </table>

</body>
</html>
```

### 7.2.2 - Convertion

```
<html>

<head>
<title>Seahorse's examples - Parsing</title>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<script type="text/javascript" src="js/seahorse-1.2.js"></script>

</head>
<body>

   <table>
      <tr>
         <td>Seahorse.parseNumber("1,234.5", '.', ',')</td>
         <td><img src="img/icon-arrow-right.png" title="=" alt="="/></td>
         <td>
            <script type="text/javascript">
               document.write(Seahorse.parseNumber("1,234.5", '.', ','));
            </script>
         </td>
      </tr>
      <tr>
         <td>Seahorse.parseInteger("1,234", ',')</td>
         <td><img src="img/icon-arrow-right.png" title="=" alt="="/></td>
         <td>
            <script type="text/javascript">
               document.write(Seahorse.parseInteger("1,234", ','));
            </script>
         </td>
      </tr>
      <tr>
         <td>Seahorse.parseIPv4("192.168.0.1")</td>
         <td><img src="img/icon-arrow-right.png" title="=" alt="="/></td>
         <td>
            <script type="text/javascript">
               document.write(Seahorse.parseIPv4("192.168.0.1"));
            </script>
         </td>
      </tr>
      <tr>
         <td>Seahorse.parseIPv6("11:22:33:44:55:66:77:88")</td>
         <td><img src="img/icon-arrow-right.png" title="=" alt="="/></td>
         <td>
            <script type="text/javascript">
               document.write(Seahorse.parseIPv6("11:22:33:44:55:66:77:88"));
            </script>
         </td>
      </tr>
```

```
      <tr>
         <td>Seahorse.parseDate("31/07/2010","dd/mm/yyyy")</td>
         <td><img src="img/icon-arrow-right.png" title="=" alt="="/></td>
         <td>
            <script type="text/javascript">
               document.write(Seahorse.parseDate("31/07/2010","dd/mm/yyyy"));
            </script>
         </td>
      </tr>
      <tr>
         <td>Seahorse.parseTime("02:05:33","hh:mm:ss")</td>
         <td><img src="img/icon-arrow-right.png" title="=" alt="="/></td>
         <td>
            <script type="text/javascript">
               document.write(Seahorse.parseTime("02:05:33","hh:mm:ss"));
            </script>
         </td>
      </tr>
   </table>

</body>
</html>
```

### 7.2.3 - Comportements

```
<html>

<head>
<title>Seahorse's examples - Parsing</title>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">

<style type="text/css">
   .ok { background-color:#f0fff0; }
   .error { background-color:#fff0f0; }
</style>

<script type="text/javascript" src="js/seahorse-1.2.js"></script>
<script type="text/javascript">
   function init()
      {
      Seahorse.number( "number1", {"decimalCharacter": '.', "groupingCharacter" : ','},
{ "okClass": "ok", "errorClass": "error" } );
      document.getElementById("number1").seahorse.verify();

      Seahorse.integer( "integer1", {"groupingCharacter" : ','}, { "okClass": "ok",
"errorClass": "error" } );
      document.getElementById("integer1").seahorse.verify();

      Seahorse.date( "date1", {"format" : 'dd/mm/yyyy', 'autofill': true}, { "okClass":
"ok", "errorClass": "error" } );
      document.getElementById("date1").seahorse.verify();

      Seahorse.time( "time1", {"format" : 'hh:mm:ss', 'autofill': true}, { "okClass":
"ok", "errorClass": "error" } );
      document.getElementById("time1").seahorse.verify();

      Seahorse.ipAddress( "ipv41", {"version" : 4 }, { "okClass": "ok", "errorClass":
"error" } );
      document.getElementById("ipv41").seahorse.verify();

      Seahorse.email( "email1", {}, { "okClass": "ok", "errorClass": "error" } );
      document.getElementById("email1").seahorse.verify();

      Seahorse.http( "http1", {}, { "okClass": "ok", "errorClass": "error" } );
      document.getElementById("http1").seahorse.verify();

      Seahorse.ftp( "ftp1", {}, { "okClass": "ok", "errorClass": "error" } );
      document.getElementById("ftp1").seahorse.verify();
```

```
     Seahorse.text( "text1", {}, { "okClass": "ok", "errorClass": "error" } );
     document.getElementById("text1").seahorse.verify();

     Seahorse.alphabetical( "alphabetical1", {}, { "okClass": "ok", "errorClass": "error"
} );
     document.getElementById("alphabetical1").seahorse.verify();

     Seahorse.alphanumeric( "alphanumeric1", {}, { "okClass": "ok", "errorClass": "error"
} );
     document.getElementById("alphanumeric1").seahorse.verify();

     Seahorse.numeric( "numeric1", {}, { "okClass": "ok", "errorClass": "error" } );
     document.getElementById("numeric1").seahorse.verify();
     }
</script>

</head>
<body onload="javascript:init();">

   <table>
     <tr>
        <td>number()</td>
        <td><input type="text" id="number1" value="1,000.00"/></td>
     </tr>
     <tr>
        <td>integer()</td>
        <td><input type="text" id="integer1" value="1,000"/></td>
     </tr>
     <tr>
        <td>date()</td>
        <td><input type="text" id="date1" value="31/12/2000"/></td>
     </tr>
     <tr>
        <td>time()</td>
        <td><input type="text" id="time1" value="23:59:59"/></td>
     </tr>
     <tr>
        <td>ipAddress()</td>
        <td><input type="text" id="ipv41" value="192.168.0.1"/></td>
     </tr>
     <tr>
        <td>email()</td>
        <td><input type="text" id="email1" value="test@server.com"/></td>
     </tr>
     <tr>
        <td>http()</td>
        <td><input type="text" id="http1" value="http://www.test.com"/></td>
     </tr>
     <tr>
        <td>ftp()</td>
        <td><input type="text" id="ftp1" value="ftp://ftp.test.com"/></td>
     </tr>
     <tr>
        <td>text()</td>
        <td><input type="text" id="text1" value="Text"/></td>
     </tr>
     <tr>
        <td>alphanumeric()</td>
        <td><input type="text" id="alphanumeric1" value="Alphanumeric123"/></td>
     </tr>
     <tr>
        <td>alphabetical()</td>
        <td><input type="text" id="alphabetical1" value="Alphabetical"/></td>
     </tr>
     <tr>
        <td>numeric()</td>
        <td><input type="text" id="numeric1" value="1234"/></td>
```

```
      </tr>
   </table>

</body>
</html>
```

### 7.2.4 - jQuery

```
<html>

<head>
<title>Seahorse's examples - Parsing</title>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">

<style type="text/css">
   .ok { background-color:#f0fff0; }
   .error { background-color:#fff0f0; }
</style>

<script type="text/javascript" src="js/jquery-1.4.2.js"></script>
<script type="text/javascript" src="js/seahorse-1.2.js"></script>
<script type="text/javascript" src="js/seahorse.jquery-1.2.js"></script>
<script type="text/javascript">
   function init()
      {
      jQuery(".number").seaBehavior(       "number",       {"decimalCharacter":       '.',
"groupingCharacter" : ','}, { "okClass": "ok", "errorClass": "error" } );
      jQuery(".number").seaVerify();

      jQuery(".integer").seaBehavior("integer", {"groupingCharacter" : ','}, { "okClass":
"ok", "errorClass": "error" } );
      jQuery(".integer").seaVerify();

      jQuery(".date").seaBehavior( "date", {"format" : 'dd/mm/yyyy', 'autofill': true},
{ "okClass": "ok", "errorClass": "error" } );
      jQuery(".date").seaVerify();

      jQuery(".time").seaBehavior( "time", {"format" : 'hh:mm:ss', 'autofill': true},
{ "okClass": "ok", "errorClass": "error" } );
      jQuery(".time").seaVerify();

      jQuery(".ipAddress")
         .seaBehavior( "ipAddress", {"version" : 4 }, { "okClass": "ok", "errorClass":
"error" } )
         .seaVerify();

      jQuery(".email")
         .seaBehavior( "email", {}, { "okClass": "ok", "errorClass": "error" } )
         .seaVerify();

      jQuery(".http")
         .seaBehavior( "http", {}, { "okClass": "ok", "errorClass": "error" } )
         .seaVerify();

      jQuery(".ftp")
         .seaBehavior( "ftp", {}, { "okClass": "ok", "errorClass": "error" } )
         .seaVerify();

      jQuery(".text")
         .seaBehavior( "text", {}, { "okClass": "ok", "errorClass": "error" } )
         .seaVerify();

      jQuery(".alphabetical")
         .seaBehavior( "alphabetical", {}, { "okClass": "ok", "errorClass": "error" } )
         .seaVerify();

      jQuery(".alphanumeric")
         .seaBehavior( "alphanumeric", {}, { "okClass": "ok", "errorClass": "error" } )
```

```
        .seaVerify();

     jQuery(".numeric")
        .seaBehavior( "numeric", {}, { "okClass": "ok", "errorClass": "error" } )
        .seaVerify();
     }
</script>

</head>
<body onload="javascript:init();">

   <table>
     <tr>
        <td>number()</td>
        <td><input type="text" class="number" value="1,000.00"/></td>
     </tr>
     <tr>
        <td>integer()</td>
        <td><input type="text" class="integer" value="1,000"/></td>
     </tr>
     <tr>
        <td>date()</td>
        <td><input type="text" class="date" value="31/12/2000"/></td>
     </tr>
     <tr>
        <td>time()</td>
        <td><input type="text" class="time" value="23:59:59"/></td>
     </tr>
     <tr>
        <td>ipAddress()</td>
        <td><input type="text" class="ipAddress" value="192.168.0.1"/></td>
     </tr>
     <tr>
        <td>email()</td>
        <td><input type="text" class="email" value="test@server.com"/></td>
     </tr>
     <tr>
        <td>http()</td>
        <td><input type="text" class="http" value="http://www.test.com"/></td>
     </tr>
     <tr>
        <td>ftp()</td>
        <td><input type="text" class="ftp" value="ftp://ftp.test.com"/></td>
     </tr>
     <tr>
        <td>text()</td>
        <td><input type="text" class="text" value="Text"/></td>
     </tr>
     <tr>
        <td>alphanumeric()</td>
        <td><input type="text" class="alphanumeric" value="Alphanumeric123"/></td>
     </tr>
     <tr>
        <td>alphabetical()</td>
        <td><input type="text" class="alphabetical" value="Alphabetical"/></td>
     </tr>
     <tr>
        <td>numeric()</td>
        <td><input type="text" class="numeric" value="1234"/></td>
     </tr>
   </table>

</body>
</html>
```

## 7.3 - Texte juridique intégral de la licence

LEGAL CODE

**Paternité - Pas d'Utilisation Commerciale - Pas de Modification 2.0 (France)**

Creative Commons n'est pas un cabinet d'avocats et ne fournit pas de services de conseil juridique. La distribution de la présente version de ce contrat ne crée aucune relation juridique entre les parties au contrat présenté ci-après et Creative Commons. Creative Commons fournit cette offre de contrat-type en l'état, à seule fin d'information. Creative Commons ne saurait être tenu responsable des éventuels préjudices résultant du contenu ou de l'utilisation de ce contrat.

*Contrat*

L'Oeuvre (telle que définie ci-dessous) est mise à disposition selon les termes du présent contrat appelé Contrat Public Creative Commons (dénommé ici « CPCC » ou « Contrat »). L'Oeuvre est protégée par le droit de la propriété littéraire et artistique (droit d'auteur, droits voisins, droits des producteurs de bases de données) ou toute autre loi applicable. Toute utilisation de l'Oeuvre autrement qu'explicitement autorisée selon ce Contrat ou le droit applicable est interdite.

L'exercice sur l'Oeuvre de tout droit proposé par le présent contrat vaut acceptation de celui-ci. Selon les termes et les obligations du présent contrat, la partie Offrante propose à la partie Acceptante l'exercice de certains droits présentés ci-après, et l'Acceptant en approuve les termes et conditions d'utilisation.

**1. Définitions**

a. **« Oeuvre »** : oeuvre de l'esprit protégeable par le droit de la propriété littéraire et artistique ou toute loi applicable et qui est mise à disposition selon les termes du présent Contrat.

b. **« Oeuvre dite Collective »** : une oeuvre dans laquelle l'oeuvre, dans sa forme intégrale et non modifiée, est assemblée en un ensemble collectif avec d'autres contributions qui constituent en elles-mêmes des oeuvres séparées et indépendantes. Constituent notamment des Oeuvres dites Collectives les publications périodiques, les anthologies ou les encyclopédies. Aux termes de la présente autorisation, une oeuvre qui constitue une Oeuvre dite Collective ne sera pas considérée comme une Oeuvre dite Dérivée (telle que définie ci-après).

c. **« Oeuvre dite Dérivée »** : une oeuvre créée soit à partir de l'Oeuvre seule, soit à partir de l'Oeuvre et d'autres oeuvres préexistantes. Constituent notamment des Oeuvres dites Dérivées les traductions, les arrangements musicaux, les adaptations théâtrales, littéraires ou cinématographiques, les enregistrements sonores, les reproductions par un art ou un procédé quelconque, les résumés, ou toute autre forme sous laquelle l'Oeuvre puisse être remaniée, modifiée, transformée ou adaptée, à l'exception d'une oeuvre qui constitue une Oeuvre dite Collective. Une Oeuvre dite Collective ne sera pas considérée comme une Oeuvre dite Dérivée aux termes du présent Contrat. Dans le cas où l'Oeuvre serait une composition musicale ou un enregistrement sonore, la synchronisation de l'oeuvre avec une image animée sera considérée comme une Oeuvre dite Dérivée pour les propos de ce Contrat.

d. **« Auteur original »** : la ou les personnes physiques qui ont créé l'Oeuvre.

e. **« Offrant »** : la ou les personne(s) physique(s) ou morale(s) qui proposent la mise à disposition de l'Oeuvre selon les termes du présent Contrat.

f. **« Acceptant »** : la personne physique ou morale qui accepte le présent contrat et exerce des droits sans en avoir violé les termes au préalable ou qui a reçu l'autorisation expresse de l'Offrant d'exercer des droits dans le cadre du présent contrat malgré une précédente violation de ce contrat.

**2. Exceptions aux droits exclusifs.** Aucune disposition de ce contrat n'a pour intention de réduire, limiter ou restreindre les prérogatives issues des exceptions aux droits, de l'épuisement des droits ou

d'autres limitations aux droits exclusifs des ayants droit selon le droit de la propriété littéraire et artistique ou les autres lois applicables.

**3. Autorisation.** Soumis aux termes et conditions définis dans cette autorisation, et ceci pendant toute la durée de protection de l'Oeuvre par le droit de la propriété littéraire et artistique ou le droit applicable, l'Offrant accorde à l'Acceptant l'autorisation mondiale d'exercer à titre gratuit et non exclusif les droits suivants :

a. eproduire l'Oeuvre, incorporer l'Oeuvre dans une ou plusieurs Oeuvres dites Collectives et reproduire l'Oeuvre telle qu'incorporée dans lesdites Oeuvres dites Collectives;

b. distribuer des exemplaires ou enregistrements, présenter, représenter ou communiquer l'Oeuvre au public par tout procédé technique, y compris incorporée dans des Oeuvres Collectives;

c. lorsque l'Oeuvre est une base de données, extraire et réutiliser des parties substantielles de l'Oeuvre.

Les droits mentionnés ci-dessus peuvent être exercés sur tous les supports, médias, procédés techniques et formats. Les droits ci-dessus incluent le droit d'effectuer les modifications nécessaires techniquement à l'exercice des droits dans d'autres formats et procédés techniques. L'exercice de tous les droits qui ne sont pas expressément autorisés par l'Offrant ou dont il n'aurait pas la gestion demeure réservé, notamment les mécanismes de gestion collective obligatoire applicables décrits à l'article 4(d).

**4. Restrictions.** L'autorisation accordée par l'article 3 est expressément assujettie et limitée par le respect des restrictions suivantes :

a. L'Acceptant peut reproduire, distribuer, représenter ou communiquer au public l'Oeuvre y compris par voie numérique uniquement selon les termes de ce Contrat. L'Acceptant doit inclure une copie ou l'adresse Internet (Identifiant Uniforme de Ressource) du présent Contrat à toute reproduction ou enregistrement de l'Oeuvre que l'Acceptant distribue, représente ou communique au public y compris par voie numérique. L'Acceptant ne peut pas offrir ou imposer de conditions d'utilisation de l'Oeuvre qui altèrent ou restreignent les termes du présent Contrat ou l'exercice des droits qui y sont accordés au bénéficiaire. L'Acceptant ne peut pas céder de droits sur l'Oeuvre. L'Acceptant doit conserver intactes toutes les informations qui renvoient à ce Contrat et à l'exonération de responsabilité. L'Acceptant ne peut pas reproduire, distribuer, représenter ou communiquer au public l'Oeuvre, y compris par voie numérique, en utilisant une mesure technique de contrôle d'accès ou de contrôle d'utilisation qui serait contradictoire avec les termes de cet Accord contractuel. Les mentions ci-dessus s'appliquent à l'Oeuvre telle qu'incorporée dans une Oeuvre dite Collective, mais, en dehors de l'Oeuvre en elle-même, ne soumettent pas l'Oeuvre dite Collective, aux termes du présent Contrat. Si l'Acceptant crée une Oeuvre dite Collective, à la demande de tout Offrant, il devra, dans la mesure du possible, retirer de l'Oeuvre dite Collective toute référence au dit Offrant, comme demandé. Si l'Acceptant crée une Oeuvre dite Collective, à la demande de tout Auteur, il devra, dans la mesure du possible, retirer de l'Oeuvre dite Collective toute référence au dit Auteur, comme demandé.

b. L'Acceptant ne peut exercer aucun des droits conférés par l'article 3 avec l'intention ou l'objectif d'obtenir un profit commercial ou une compensation financière personnelle. L'échange de l'Oeuvre avec d'autres Oeuvres protégées par le droit de la propriété littéraire et artistique par le partage électronique de fichiers, ou par tout autre moyen, n'est pas considéré comme un échange avec l'intention ou l'objectif d'un profit commercial ou d'une compensation financière personnelle, dans la mesure où aucun paiement ou compensation financière n'intervient en relation avec l'échange d'Oeuvres protégées.

c. Si l'Acceptant reproduit, distribue, représente ou communique l'Oeuvre au public, y compris par voie numérique, il doit conserver intactes toutes les informations sur le régime des droits et en attribuer la paternité à l'Auteur Original, de manière raisonnable au regard au médium ou au moyen utilisé. Il doit communiquer le nom de l'Auteur Original ou son éventuel pseudonyme s'il est indiqué ; le titre de l'Oeuvre Originale s'il est indiqué ; dans la mesure du possible, l'adresse Internet ou Identifiant Uniforme de Ressource (URI), s'il existe, spécifié par l'Offrant comme associé à l'Oeuvre, à moins que cette adresse ne renvoie pas aux informations légales (paternité et conditions d'utilisation de l'Oeuvre). Ces obligations d'attribution de paternité doivent être exécutées de manière raisonnable. Cependant, dans le cas d'une Oeuvre dite Collective, ces informations doivent, au minimum, apparaître à la place et de manière aussi visible que celles à laquelle apparaissent les informations de même nature.

d. Dans le cas où une utilisation de l'Oeuvre serait soumise à un régime légal de gestion collective obligatoire, l'Offrant se réserve le droit exclusif de collecter ces redevances par l'intermédiaire de

la société de perception et de répartition des droits compétente. Sont notamment concernés la radiodiffusion et la communication dans un lieu public de phonogrammes publiés à des fins de commerce, certains cas de retransmission par câble et satellite, la copie privée d'Oeuvres fixées sur phonogrammes ou vidéogrammes, la reproduction par reprographie.

**5. Garantie et exonération de responsabilité**

a. En mettant l'Oeuvre à la disposition du public selon les termes de ce Contrat, l'Offrant déclare de bonne foi qu'à sa connaissance et dans les limites d'une enquête raisonnable :

   I. L'Offrant a obtenu tous les droits sur l'Oeuvre nécessaires pour pouvoir autoriser l'exercice des droits accordés par le présent Contrat, et permettre la jouissance paisible et l'exercice licite de ces droits, ceci sans que l'Acceptant n'ait aucune obligation de verser de rémunération ou tout autre paiement ou droits, dans la limite des mécanismes de gestion collective obligatoire applicables décrits à l'article 4(e);

   II. L'Oeuvre n'est constitutive ni d'une violation des droits de tiers, notamment du droit de la propriété littéraire et artistique, du droit des marques, du droit de l'information, du droit civil ou de tout autre droit, ni de diffamation, de violation de la vie privée ou de tout autre préjudice délictuel à l'égard de toute tierce partie.

a. A l'exception des situations expressément mentionnées dans le présent Contrat ou dans un autre accord écrit, ou exigées par la loi applicable, l'Oeuvre est mise à disposition en l'état sans garantie d'aucune sorte, qu'elle soit expresse ou tacite, y compris à l'égard du contenu ou de l'exactitude de l'Oeuvre.

**6. Limitation de responsabilité.** A l'exception des garanties d'ordre public imposées par la loi applicable et des réparations imposées par le régime de la responsabilité vis-à-vis d'un tiers en raison de la violation des garanties prévues par l'article 5 du présent contrat, l'Offrant ne sera en aucun cas tenu responsable vis-à-vis de l'Acceptant, sur la base d'aucune théorie légale ni en raison d'aucun préjudice direct, indirect, matériel ou moral, résultant de l'exécution du présent Contrat ou de l'utilisation de l'Oeuvre, y compris dans l'hypothèse où l'Offrant avait connaissance de la possible existence d'un tel préjudice.

**7. Résiliation**

a. Tout manquement aux termes du contrat par l'Acceptant entraîne la résiliation automatique du Contrat et la fin des droits qui en découlent. Cependant, le contrat conserve ses effets envers les personnes physiques ou morales qui ont reçu de la part de l'Acceptant, en exécution du présent contrat, la mise à disposition d'Oeuvres dites Dérivées, ou d'Oeuvres dites Collectives, ceci tant qu'elles respectent pleinement leurs obligations. Les sections 1, 2, 5, 6 et 7 du contrat continuent à s'appliquer après la résiliation de celui-ci.

b. Dans les limites indiquées ci-dessus, le présent Contrat s'applique pendant toute la durée de protection de l'Oeuvre selon le droit applicable. Néanmoins, l'Offrant se réserve à tout moment le droit d'exploiter l'Oeuvre sous des conditions contractuelles différentes, ou d'en cesser la diffusion; cependant, le recours à cette option ne doit pas conduire à retirer les effets du présent Contrat (ou de tout contrat qui a été ou doit être accordé selon les termes de ce Contrat), et ce Contrat continuera à s'appliquer dans tous ses effets jusqu'à ce que sa résiliation intervienne dans les conditions décrites ci-dessus.

**8. Divers**

a. A chaque reproduction ou communication au public par voie numérique de l'Oeuvre ou d'une Oeuvre dite Collective par l'Acceptant, l'Offrant propose au bénéficiaire une offre de mise à disposition de l'Oeuvre dans des termes et conditions identiques à ceux accordés à la partie Acceptante dans le présent Contrat.

b. La nullité ou l'inapplicabilité d'une quelconque disposition de ce Contrat au regard de la loi applicable n'affecte pas celle des autres dispositions qui resteront pleinement valides et applicables. Sans action additionnelle par les parties à cet accord, lesdites dispositions devront être interprétées dans la mesure minimum nécessaire à leur validité et leur applicabilité.

c. Aucune limite, renonciation ou modification des termes ou dispositions du présent Contrat ne pourra être acceptée sans le consentement écrit et signé de la partie compétente.

d. Ce Contrat constitue le seul accord entre les parties à propos de l'Oeuvre mise ici à disposition. Il

n'existe aucun élément annexe, accord supplémentaire ou mandat portant sur cette Oeuvre en dehors des éléments mentionnés ici. L'Offrant ne sera tenu par aucune disposition supplémentaire qui pourrait apparaître dans une quelconque communication en provenance de l'Acceptant. Ce Contrat ne peut être modifié sans l'accord mutuel écrit de l'Offrant et de l'Acceptant.

e. Le droit applicable est le droit français.

Creative Commons n'est pas partie à ce Contrat et n'offre aucune forme de garantie relative à l'Oeuvre. Creative Commons décline toute responsabilité à l'égard de l'Acceptant ou de toute autre partie, quel que soit le fondement légal de cette responsabilité et quel que soit le préjudice subi, direct, indirect, matériel ou moral, qui surviendrait en rapport avec le présent Contrat. Cependant, si Creative Commons s'est expressément identifié comme Offrant pour mettre une Oeuvre à disposition selon les termes de ce Contrat, Creative Commons jouira de tous les droits et obligations d'un Offrant.

A l'exception des fins limitées à informer le public que l'Oeuvre est mise à disposition sous CPCC, aucune des parties n'utilisera la marque « Creative Commons » ou toute autre indication ou logo afférent sans le consentement préalable écrit de Creative Commons. Toute utilisation autorisée devra être effectuée en conformité avec les lignes directrices de Creative Commons à jour au moment de l'utilisation, telles qu'elles sont disponibles sur son site Internet ou sur simple demande.

Creative Commons peut être contacté à http://creativecommons.org/.